UNIVERSIDADE DA CORUÑA

Departamento de Electrónica y Sistemas

Arquitectura Distribuida por Delegación para la Gestión de Alarmas en Redes de Telecomunicaciones

Víctor M. Carneiro Díaz

Director: Dr. Angel Viña Castiñeiras

A Coruña, Julio 1998

UNIVERSIDADE DA CORUÑA
Dpto. Electrónica y Sistemas

Arquitectura Distribuida por Delegación para la Gestión de Alarmas en Redes de Telecomunicaciones

Víctor M. Carneiro Díaz

Director: Dr. Angel Viña Castiñeiras
A Coruña, Julio 1998

Dr. Ángel Viña Castiñeiras,

Catedrático de Ingeniería Telemática del
Dpto. de Electrónica y Sistemas de la
Universidad de A Coruña.

CERTIFICA:

Que la memoria titulada "*Arquitectura Distribuida por Delegación para la Gestión de Alarmas en Redes de Telecomunicaciones*", ha sido realizada por D. Víctor Manuel Carneiro Díaz bajo mi dirección en el Departamento de Electrónica y Sistemas de la Universidad de A Coruña y concluye la tesis que presenta para optar al grado de Doctor en Informática.

A Coruña, 23 de Marzo de 1998.

Fdo.: D José M. Domínguez Legaspi
Director del Dpto. de Electrónica y Sistemas

Fdo.: Dr. Ángel Viña Castiñeiras
Director de la Tesis Doctoral

Resumen

Arquitectura Distribuida por Delegación para la

Gestión de Alarmas en Redes de Telecomunicaciones.

Víctor M. Carneiro Díaz

Las complejas y modernas redes de telecomunicaciones, orientadas en muchos casos a garantizar niveles de servicio por contrato, necesitan de potentes herramientas de gestión de alarmas. La gestión eficiente de alarmas constituye la base para la implantación del resto de áreas funcionales de gestión de red. Los tradicionales sistemas de gestión heredan las limitaciones de los modelos centralizados y la ausencia de riqueza semántica en la información acerca de los eventos generados por los nodos de red. Esta disertación presenta una arquitectura distribuida de gestión de alarmas basada en el paradigma de la *Gestión por Delegación* (MbD) que solventa estas deficiencias, aportando al operador un entorno integrado y homogéneo en el que pueden coexistir alarmas de diversos protocolos. La validez de esta arquitectura ha sido analizada mediante el desarrollo de un sistema de gestión de alarmas independiente de la plataforma y de la localización de los distintos componentes que lo constituyen. Para el manejo de alarmas se utiliza un formato uniforme de acuerdo con la norma ITU-T X.721. Este formato, de gran riqueza semántica, puede ser utilizado para la gestión de equipos con limitaciones en el contenido de la información que generan los eventos, incorporando datos estáticos acerca de los equipos gestionados. El sistema resultante presenta al programador un entorno flexible, modular y robusto que le permite dinámicamente aumentar la funcionalidad de los agentes sin modificar la codificación de ninguno de ellos.

Contenidos

Agradecimientos

Esta tesis no podría haber sido escrita sin el apoyo de otra gente. Quisiera expresar mi agradecimiento por esta ayuda, en particular a las siguientes personas.

En primer lugar, agradecer a mi director de tesis, D. Angel Viña, la oportunidad que me ha brindado al introducirme en el innovador mundo de la gestión de red. Su experiencia y comentarios acerca del enfoque al afrontar una disertación de este tipo, me han guiado enormemente.

Quisiera también agradecer al personal docente e investigador del Dpto. de Electrónica y Sistemas de la Universidad de A Coruña, las facilidades que siempre me han dispensado, y en mayor medida, al grupo de gestión de redes que dirige el profesor Viña, por el acogedor y estimulante ambiente de trabajo que han creado.

A Fernando Bellas por tener siempre a punto la respuesta a cualquier tema relacionado con los sistemas distribuidos, lo mismo que Carmen Guerrero con respecto a TMN. Agradecer también, a Javier Coego, haber soportado heroicamente mis primeras disertaciones. A Juan Carlos Agrelo, Francisco Muñoz y Pedro Ignacio Dorado por haberme introducido en la problemática asociada a la gestión de fallos y trasladarme sus experiencias en este campo.

Al personal de Telefónica Sistemas y Unión Fenosa por apostar por este grupo de investigadores para el desarrollo de parte de su sistema de gestión. Lo que ha hecho posible el trabajo en campos tecnológicamente poco explorados por empresas españolas.

Finalmente quisiera agradecer a mi familia y especialmente a mis padres haberme dado la oportunidad de realizar mis estudios. Especial gratitud para mi mujer Rosa, cuyo continuo apoyo y paciencia ha sido indispensable para poder completar este trabajo.

Prefacio

Una de las áreas funcionales de mayor interés dentro de la gestión de red es la gestión de fallos. Su importancia se debe a sus interacciones con el resto de funcionalidades de gestión y a que en las actuales redes de telecomunicaciones es vital, para garantizar por contrato un determinado nivel de servicio, disponer de un control eficiente de los fallos que surgen en los elementos de red. Algunos de los problemas con lo que el operador se encuentra son los fallos en los dispositivos, rendimientos ineficientes, inseguridad frente a posibles ataques y fallos en la contabilidad/utilización de los recursos que ha de solucionar de manera automatizada con un potente gestor de fallos. Dentro de la gestión de fallos, una de las tareas más importantes es el tratamiento o gestión de alarmas, objeto del presente trabajo.

Sin embargo, los tradicionales sistemas de gestión de alarmas, generalmente diseñados como arquitecturas centralizadas, presentan limitaciones derivadas de su estructura y de la falta de riqueza semántica en la información que manejan.

El primer grupo de limitaciones de los tradicionales sistemas de gestión de alarmas es que, si bien alertan de la ocurrencia de un fallo, no proporcionan información adicional que permita identificar el origen del mismo. Muchos de los protocolos propietarios de gestión y algunos de los estándares como el *Simple Network Management Protocol* (SNMP), no han sido diseñados para transportar este tipo de información en sus *Protocol Data Units* (PDUs). Por otro lado, la familia de estándares OSI incorpora más información en sus primitivas, pero se queda corta en ciertos aspectos semánticos. Tradicionalmente se ha solventado esta limitación con la utilización de robustas bases de datos orientadas a objetos, que permiten almacenar información estática adicional acerca de las alarmas generadas en una red. Esta información adicional es conocida previamente por el operador y puede ser el resultado de la experiencia del mismo, de las especificaciones técnicas del dispositivo a gestionar, de la experiencia con la gestión de un determinado tipo de alarmas, etc. En cualquier caso, permite aumentar el conocimiento que el operador tiene acerca de la identificación y localización del fallo.

Otro aspecto a contemplar es la cohabitación entre protocolos propietarios y estándares. En la mayoría de los casos esta es inviable. El operador recibe información de las alarmas en diferentes formatos, lo que dificulta enormemente la comprensión y tratamiento de las mismas. ITU-T propone en su estándar X.721 [ITU-T X721] el modelo de información a seguir para el intercam-

bio y registro de notificaciones entre entidades de gestión. El uso de este estándar permite construir sistemas con un conjunto uniforme de tipos de notificaciones y con parámetros normalizados.

Otro grupo de limitaciones de los actuales sistemas de gestión de alarmas, proceden de su arquitectura centralizada, utilizada tradicionalmente por razones de baja capacidad de los recursos de red y homogeneidad de los equipos. El enfoque centralizado de gestión separa lógica y físicamente las aplicaciones de los datos que maneja, así como los recursos de los procesos que los controlan. Los agentes actúan como simples almacenes recolectores y emisores de información a los gestores que en ciertos intervalos se ven desbordados por la información recibida. En un entorno de gestión amplio y heterogéneo, la no escalabilidad de estas plataformas hace que el intercambio de datos entre agentes y gestor pueda exceder las capacidades de la propia red. Si el número de dispositivos, el número de variables a manejar y la velocidad de la red se incrementa considerablemente, el sistema se vuelve rápidamente inmanejable. Además, durante un periodo de fallo de la red, la política de enviar todas las notificaciones hacia un gestor central, no hace sino empeorar la situación debido al retardo que estas comunicaciones introducen en la propia red.

En esta disertación se introduce una arquitectura distribuida que permite solventar, en gran medida, estas limitaciones. Ha sido desarrollada basándose en el paradigma de la gestión por delegación o *Management by Delegation* (MbD en adelante), introducida en [Gold, 95]. Este paradigma utiliza el concepto de proceso elástico para crear un entorno de operación flexible y totalmente distribuido. Un proceso de este tipo, es un programa que puede extender o contraer su funcionalidad en tiempo de ejecución. Para esto cuenta con una arquitectura de soporte a la delegación y de un servicio de intercambio de primitivas entre diversos procesos elásticos.

La arquitectura ha sido diseñada centrándose en los objetivos de facilitar al operador la integración de alarmas de distintos orígenes y con distintos protocolos, utilizar un modelo único de información para todos los tipos de notificaciones y mejorar la operación y mantenimiento del sistema de gestión.

La arquitectura propuesta cuenta con dos tipos de nodos de gestión:

- (a) *nodos especializados*, encargados de almacenar y controlar ciertas rutinas de gestión de alarmas, como filtros, adaptadores de protocolos, rutinas de *polling*, etc.
- (b) *nodos generales*, que son procesos elásticos encargados de la gestión de uno o varios elementos de red, pueden recibir códigos de los nodos especializados y modificar su funcionalidad para adaptarla a las necesidades de gestión.

Todo el sistema está gobernado por un gestor de configuración, que aporta la funcionalidad necesaria para mantener la coherencia y cooperación entre los diversos nodos del sistema. Para acceder a este sistema distribuido se ha diseñado un visor de alarmas multiusuario, cuya ejecución es independiente de la localización del resto de nodos del sistema. También se han implementado los módulos necesarios para permitir el intercambio de primitivas y/o código entre los distintos nodos que forman el sistema. Para esta implementación, se ha utilizado el lenguaje Java (con sus funcionalidades RMI y JDBC). El sistema resultante muestra las ventajas de la utilización de la arquitectura propuesta en la gestión de alarmas.

Esta memoria está estructurada de la siguiente forma:

Capítulo 1: Realiza una descripción pormenorizada de la problemática asociada al área funcional de la gestión de fallos (en la que está enmarcada la gestión de alarmas). Se introducen los conceptos básicos que deben conocerse a la hora de afrontar el diseño o manejo de un sistema de

gestión de alarmas. Las principales funcionalidades de la gestión de fallos son: identificar la ocurrencia de un fallo, localizarlo, aislarlo y, si es posible, corregirlo. Su objetivo es incrementar la disponibilidad de la red, proporcionando al ingeniero de red ayudas para detectar y corregir las posibles anomalías o errores que se produzcan en la red.

Capítulo 2: En este capítulo se analizan y evalúan una serie de modelos de gestión que actualmente son utilizados en distintas áreas de la gestión de red. Su estudio se basa en unos criterios cualitativos que permiten validar la utilidad de cada uno de ellos para la construcción de un sistema de gestión de alarmas. El análisis de estos modelos ha servido de base para el desarrollo de la arquitectura propuesta en esta tesis.

Capítulo 3: Presenta las modificaciones introducidas en el paradigma de la Gestión por Delegación (MbD), así como el diseño de una arquitectura distribuida que solventa la mayoría de las limitaciones encontradas en el estudio de los modelos actuales. La arquitectura resultante aporta flexibilidad de manejo y expansión, distribución del procesamiento (la información es tratada allí donde se produce) así como tratamiento y registro uniforme de las alarmas. También facilita la integración de alarmas generadas por cualquier protocolo de gestión existente.

Capítulo 4: En este capítulo se describe una implementación de la arquitectura presentada utilizando el lenguaje de programación Java. Se describe el diseño y desarrollo de cada uno de los módulos que constituyen la arquitectura, analizando la validez de la misma para la gestión de alarmas. También se describen las nuevas funcionalidades que adquiere el sistema con la utilización de un lenguaje de programación independiente de la plataforma y orientado a objetos.

Capítulo 5: Se presenta un sumario de conclusiones y posibles trabajos futuros sobre la base de la arquitectura desarrollada.

También se incorpora un apéndice que sirve como guía de usuario para aquellas personas que deseen utilizar el sistema de gestión desarrollado, próximamente de libre distribución para la comunidad científica internacional.

Contribuciones

Esta tesis presenta una arquitectura distribuida, basada en la Gestión por Delegación (MbD), para solventar la mayoría de las limitaciones encontradas en los actuales sistemas de gestión, en lo que a tratamiento de alarmas se refiere. Concretamente esta tesis incluye las siguientes contribuciones:

- Análisis y evaluación de un conjunto de modelos de gestión de red representativos. Se estudian las carencias, limitaciones y funcionalidades que tanto la tecnología utilizada como su arquitectura aportan al manejo de alarmas. De este estudio surgen los fundamentos de la arquitectura propuesta en este trabajo.
- Introducción de modificaciones en el paradigma de la Gestión por Delegación (MbD), para incorporar las funcionalidades necesarias para el tratamiento de alarmas. Entre otras modificaciones, se han introducido nuevas primitivas para la solicitud de código por parte de un proceso elástico, el manejo de información global de configuración, la modificación de la arquitectura en capas para incorporar módulos de nivel superior que se comuniquen con el proceso elástico de bajo nivel, etc.
- Desarrollo de una arquitectura distribuida de gestión de alarmas que utiliza un formato uniforme para el registro y notificación de eventos. Permite integrar cualquier protocolo de gestión disponible actualmente, facilitando enormemente su expansión y escalabili-

dad, mediante la incorporación de nuevas funcionalidades en forma de procesos delegados. Esta arquitectura tiene como característica especial su facilidad para adaptarse a las necesidades de procesamiento pudiendo adoptar modelos centralizados, jerárquicos o distribuidos, en base a las necesidades de tratamiento de alarmas.

- Implementación de la arquitectura diseñada utilizando el lenguaje de programación Java y sus características RMI y JDBC. El sistema resultante valida la arquitectura, aportando nuevas funcionalidades inherentes a Java, como la independencia de la plataforma y la movilidad de código.

1.- Conceptos sobre gestión de fallos

El movimiento liberalizador que tiene lugar en el área de las telecomunicaciones y de los servicios en general, se ha traducido en un crecimiento exponencial de las redes y de sus necesidades de gestión [Guerrero, 97a]. Se hace necesaria una automatización y estandarización de esta gestión, de forma que se le facilite al operador el tratamiento de la información que recibe de los diversos equipos. A continuación se resumen los pilares básicos que sustentan este crecimiento:

- El número de nodos conectados se ha incrementado enormemente debido a la reducción de precios del *hardware*. Las tradicionales salas con terminales de texto conectados a un *mainframe* se han visto sustituidas por puestos de trabajo económicos y de altas prestaciones. Esto ha generado un cambio de enfoque en cuanto a la arquitectura de las redes, pasando de un entorno de operación centralizado a uno distribuido. Este cambio ha potenciado el desarrollo de aplicaciones distribuidas que incrementen el procesamiento e inteligencia de todos los puestos de trabajo de la red.
- Debido al aumento de la capacidad de estos equipos y las aplicaciones que utilizan, se ha producido un gran incremento de las prestaciones de las redes de telecomunicación, lo que se ha traducido en mejores comunicaciones entre equipos terminales. El desarrollo del *hardware* posibilita la realización de interconexiones de alta velocidad con el consiguiente incremento exponencial de la complejidad de la arquitectura de las modernas redes. Actualmente, en una red de telecomunicaciones, conviven gran variedad de interfaces, protocolos y proveedores. El intercambio de información de gestión se hace inviable sin unas interfaces estandarizadas. Hasta el momento la adopción de soluciones propietarias ha complicado enormemente la integración de sistemas.
- Actualmente, las empresas necesitan contar con un control efectivo de los recursos de su red y contabilizar el coste de utilización de los mismos. Debido al gran número y diversidad de los actuales equipos es necesario el uso de avanzadas herramientas de gestión para llevar a cabo estas labores.
- El gran dinamismo tecnológico provoca errores en la fase de diseño de la red de comunicaciones que debe subsanarse posteriormente durante la fase operacional apoyándose en las funciones de gestión.

Todos estos aspectos incrementan la complejidad de las tareas de gestión y demandan la adopción de una gestión automatizada de los recursos de la red, lo que se realiza actualmente con plataformas de gestión de red generalmente centralizadas. Además, la gestión de red debe contemplarse siempre desde un punto de vista global, abierto y estándar, siguiendo una política de reducción de los costes de explotación, una compensación de la falta de experiencia de ciertos usuarios en el manejo de un entorno dinámico, una mejora en el tratamiento de fallos y una mayor flexibilidad de operación.

Resulta complicado dar una definición para el término gestión de red, por lo que utilizamos dos de las definiciones más aceptadas:

- *"La gestión de red incluye las funciones de operación, administración, mantenimiento y disposiciones necesarias para proporcionar, monitorizar, interpretar y controlar una red y sus servicios asociados"* [Aidarous, 94] .
- *"La gestión de red es la acción de inicialización, monitorización y modificación de operación de las funciones primarias de red, entendiendo éstas como las que soportan directamente los requerimientos del usuario"* [Bharat, 96] .

En las dos definiciones se relata el conjunto de funcionalidades y objetivos que se pretende cubrir con la gestión de red. Como se aprecia, la gestión de red no incluye únicamente la monitorización de equipos *hardware*, sino que también incluye la gestión de servicios y requerimientos de usuario, garantizando las expectativas que se pretenden de la utilización de una red de telecomunicaciones.

La gestión de fallos ha sido identificada como una de las áreas funcionales de gestión de red (denominadas comúnmente FCAPS) más importantes. Esto es debido a sus interrelaciones con otras áreas como gestión de configuración y la gestión de seguridad. Las principales funcionalidades de la gestión de fallos consisten en identificar la ocurrencia de un fallo, localizarlo, aislarlo y corregirlo (sí es posible). Su objetivo es incrementar la disponibilidad de la red, proporcionando al ingeniero de red ayudas para detectar y corregir las posibles anomalías o errores que se producen en su entorno de red. En este trabajo nos centraremos en una de las funcionalidades de mayor importancia de la gestión de fallos que es el tratamiento de alarmas.

A la hora de plantearse la realización de un sistema de gestión de fallos, y por lo tanto de un sistema de gestión de alarmas, deben tenerse en cuenta los condicionantes que se dan en una red de telecomunicaciones:

- Es difícil operar sobre una red que está fallando. Alguno de los elementos de red puede no mostrar conectividad, lo que hace inviable la operación sobre el mismo, sin embargo, esta ausencia de conectividad es una situación muy común cuando está ocurriendo un fallo.
- La detección, determinación y diagnóstico de fallos en muchas organizaciones es especialmente complejo y repercute en pérdidas económicas. Un fallo puede causar un gran decremento del rendimiento de una red de telecomunicaciones o incluso su paralización. Esta situación transitoria puede provocar la alteración del plan de trabajo de ciertas entidades, dejando en algunos casos, como el de entidades bancarias, en entredicho la seguridad de sus equipos de comunicaciones con el correspondiente perjuicio que esto supone.

- La rigidez de la instrumentación actual hace muy difícil la determinación del problema. En muchos casos, los equipos no están diseñados para llevar a cabo una gestión estandarizada, dificultando enormemente estas tareas. Otras veces, no puede llevarse a cabo telecontrol sobre el equipo, lo que impide la realización de mecanismos de acción remota y limita enormemente la utilización de herramientas de gestión.
- Los centros de control trabajan con múltiples consolas para las representaciones de los elementos de red o sistemas de gestión remotos. Es necesario planificar y establecer unos apropiados mecanismos de coordinación entre los diversos centros, así como los mecanismos de seguridad necesarios que permitan asignar la funcionalidad requerida para cada tipo de operador que se conecta al sistema de gestión.
- Ciertos fallos no son apreciables mediante una observación directa. En otros, la instrumentación del equipo no ha registrado la posibilidad de ese fallo. Muchos fallos aunque son observables, esto no es suficiente para determinar la causa principal del problema. Otras veces ocurre que se producen inconsistencias entre diferentes observaciones del mismo fallo.
- Para conocer si ha fallado o no un dispositivo es necesario extraer la información acerca del estado del mismo. Para ello existen dos métodos: esperar a que el dispositivo nos notifique el evento o crear un proceso que consulte con una cierta periodicidad (*frecuencia de polling*) el estado del dispositivo. El establecimiento de una correcta frecuencia de *polling* puede resultar una tarea crítica en el funcionamiento del sistema, dependiendo en gran medida del ancho de banda que se consume con estas interrogaciones y el número de dispositivos que es necesario interrogar.
- Una tarea fundamental en la gestión de fallos es la priorización de los mismos. Dado que no todos los fallos son críticos, es necesario identificar los más importantes. Para conseguir esto, una solución consiste en que el ingeniero de red configure por separado cada dispositivo indicando únicamente los eventos que le interesen; sin embargo, esto no es una buena solución, el sistema de gestión debe filtrar los eventos y alertar al ingeniero de red de otros eventos no especificados. La determinación de qué fallos deben ser gestionados está influenciado por el control que se quiere tener de la red y el tamaño de la red a gestionar. Cuando el tamaño o la dispersión geográfica de la red a gestionar es elevado, deben explorarse sistemas de gestión basados en arquitecturas distribuidas o jerárquicas.

El principal objetivo de la gestión de fallos es la restauración del servicio en presencia de fallos, la identificación de la causa principal del fallo y finalmente una reparación rápida y eficiente reduciendo en gran medida el coste del aislamiento y reparación del mecanismo que ha fallado. Es importante también analizar o establecer el balance deseado entre servicio y coste que permita garantizar un determinado nivel de servicio al usuario.

Algunos de los aspectos [Aidarous, 94] a tener en cuenta en el desarrollo de un sistema de gestión de fallos son los siguientes:

- ***Disponibilidad***: En gestión de fallos, la disponibilidad se caracteriza por el tiempo medio de restauración del servicio o el tiempo medio de reparación. Los avances tecnológicos han aumentado la disponibilidad de los sistemas (fibra óptica, circuitos VLSI, circuitos digitales...), sin embargo, existen condicionantes naturales (fuego, terremotos, inundaciones...) y humanos o mecánicos (errores en programas, en bases de datos, durante la inicialización de máquinas...) que influyen en la expansión de los fallos. Planificar y tener en cuenta la posibilidad de estos fallos es tarea del gestor de fallos.

- ***Restauración del servicio***: Existen dos aproximaciones, la restauración *proactiva* y la *reactiva*. Esta última se basa en una estrategia de reparación más rápida posible del elemento que ha fallado. Por otro lado, la estrategia proactiva enfatiza la utilización de arquitecturas de red redundantes que permitan que un dispositivo pueda ser aislado y reemplazado por otros equipos de las mismas características que se encuentran dispersos por la red. Otra estrategia proactiva, consiste en analizar las tendencias de la degradación y realizar reparaciones antes que se produzca una degradación seria de la calidad de servicio.
- ***Identificación de la causa principal***: Antes de proceder a la reparación de un fallo, debe identificarse la causa o causas principales del mismo. Para ello es necesario contar con unos mecanismos de diagnosis y chequeo apropiados. Durante el proceso de identificación de la causa principal pueden llevarse a cabo procesos de correlación [Jakobson, 93] que permita establecer las interrelaciones entre los eventos recibidos.
- ***Reparación de fallos***: Consiste en la asignación de trabajos, herramientas y recambios, y distribución de las mismas en la localización del fallo. Reparar el fallo, verificar que funciona adecuadamente y devolverlo al estado de disponibilidad para el servicio.
- ***Medida de la efectividad***: Aquello que no puede ser medido, no puede ser gestionado efectivamente. Es interesante mantener un histórico de fallos y actuaciones de resolución del fallo para lograr un sistema de gestión de fallos.

1.1 Tareas de Gestión de Fallos

En esta sección se realiza una introducción a las funcionalidades del área de gestión de fallos. Aunque el objetivo de este trabajo se ha centrado en mejorar aspectos relativos a la gestión de alarmas (englobada dentro de la gestión de fallos), es interesante analizar en global la problemática asociada a la misma.

En gestión de fallos deben incorporarse mecanismos para la detección, indicación, localización y restauración de fallos. Podemos clasificar estos mecanismos en las siguientes tareas [Aidarous, 94] :

- *Vigilancia de alarmas*: Consiste en la recolección, registro, análisis de la causa principal e inicio de las rutinas de localización y chequeo para verificar esos síntomas.
- *Localización de fallos*: Rutinas encargadas de determinar el origen principal del fallo.
- *Chequeo*: Una vez localizado el origen, las rutinas de chequeo se encargan de detectar y aislar el error que ha producido el fallo.
- *Restauración del servicio*: Rutinas que permitan aislar el fallo de forma que pueda restaurarse el nivel de servicio perdido.
- *Reparación*: Planificación del trabajo a realizar para reponer/reparar el elemento que ha fallado.

La figura 1.1 [Aidarous, 94] muestra la arquitectura general de un sistema de gestión de fallos.

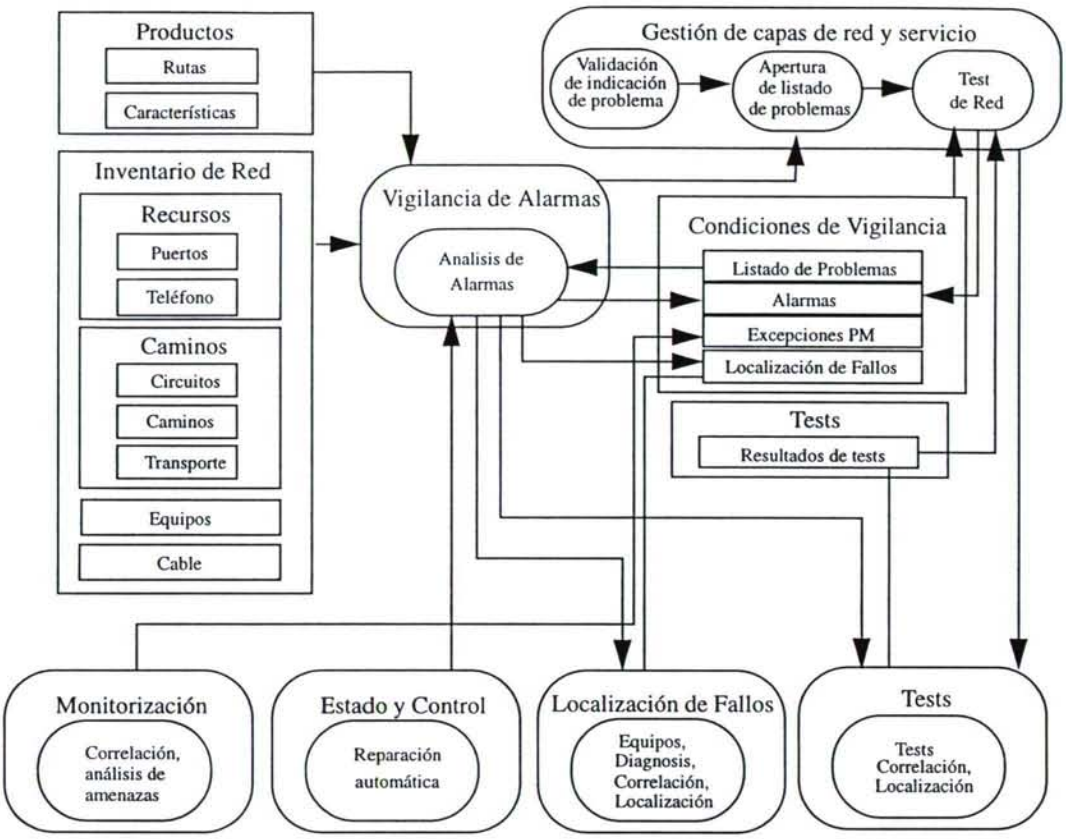

Figura 1.1. *Arquitectura de un sistema de gestión de fallos*

A continuación se describe, con mayor profundidad, cada uno de estos métodos:

1.1.1 Vigilancia de alarmas

La vigilancia de alarmas recolecta los eventos que recibe de las distintas rutinas de chequeo dispersas por la red. Recibe también información emitida por los procesos de restauración automática del gestor de configuración. Un proceso de la vigilancia de alarmas es el análisis de las mismas, lo que permite identificar qué alarmas son significativas. Estas alarmas seleccionadas son almacenadas en la base de datos de condiciones de vigilancia y enviadas, tanto al módulo de localización de fallos como al de *test*. Esto permite al proceso de análisis identificar por inferencia la causa principal del fallo y su localización específica en la base de datos, iniciando las rutinas de chequeo necesarias que permitan verificar estos datos. Eventualmente utiliza el inventario de red para conocer las dependencias de interrelación internas de un equipo o entre equipos.

Por tanto, la vigilancia de alarmas comienza con la detección del fallo, entendido como la persistencia de una condición de error en un determinado equipo de la red, y se traduce en el cambio de valor de una determinada variable o atributo que está modelando una característica de ese equipo.

Comprende los siguientes mecanismos:

- ***Detección***: Muchas veces el conjunto de rutinas de chequeo es redundante con lo cual se producen muchas alarmas para un único fallo. Esto es un problema a la hora de determinar la causa principal del fallo pero puede resultar de utilidad cuando queremos asegurarnos que un fallo no pase desapercibido y pueda desencadenar un error de mayor magnitud. También hay que tener en cuenta las posibles falsas alarmas producto de inducciones temporales como puede ser una corriente eléctrica. Debe chequearse el estado anterior del fallo antes de emitir un nuevo evento para garantizar que no se produce un desbordamiento de información sobre un fallo ya notificado.
- ***Obtención de información***: Cuando se chequea un fallo, es necesario extraer información acerca de la hora en que ocurrió, el tipo de fallo, y la unidad hardware donde ha sido localizado. También debe incluirse el grado de severidad del fallo. Así, distinguiremos entre avisos, alarmas críticas, mayores o menores, dependiendo de la naturaleza del fallo. Algunos equipos no pueden emitir alarmas. Para subsanar esta limitación, se recurre al diseño de diagramas de estado cuyas transiciones de estados pueden generar alarmas.
- ***Creación de un log de alarmas***: Las alarmas recibidas deben ser almacenadas en un histórico. Cada registro de alarma tiene información acerca del evento que la generó. Este histórico de alarmas puede usarse para implementar mecanismos de correlación que permitan anidar ocurrencias de eventos. Otras veces, cuando el dispositivo que falla no puede comunicarse con el centro receptor de alarmas, el *log* almacena las secuencias de fallos producidas durante el período de no conectividad.
- ***Realización del sumario de alarmas***: Recoge el estado de una alarma en un conjunto de recursos. Para caracterizar el estado de un equipo suele hacerse en base a su estado operacional (habilitado/deshabilitado), su estado administrativo (bloqueado/desbloqueado/fuera de servicio) y su estado de uso (inactivo/activo/ocupado).
- ***Comprobaciones para la emisión de alarmas***: Cada registrador de alarmas está interesado en recibir un determinado tipo de alarmas. Las reglas que manejan la entrega de información en los recipientes son conocidas como condiciones de emisión de la alarma. Existen algoritmos para determinar a que recipiente(s) debe ser enviada una alarma.
- ***Reducción de la redundancia***: Idealmente, sólo debería existir un tipo de alarma para cada rutina de detección específica. Sin embargo, los fallos se encuentran clasificados en clases y cuando uno de estos ocurre, involucran múltiples eventos. El análisis de los tipos de chequeos y la experiencia, pueden identificar síndromes relativos a múltiples fallos. Existen algoritmos que permiten identificar estos síndromes y reducir así la emisión de alarmas redundantes.
- ***Correlación de alarmas***: Un fallo puede producir muchas alarmas en múltiples equipos distribuidos en el tiempo y en el espacio. En estos casos, debe existir un proceso que correlacione todas estas alarmas e infiera la causa principal del fallo. La utilización de técnicas de reducción de redundancia puede ayudar en estos procesos.

La reducción de redundancia, correlación y análisis puede realizarse en múltiples niveles. En cada nivel, un proceso puede acceder a datos apropiados al correspondiente dominio de gestión.

1.1.2 Localización de fallos

Una vez detectado un fallo en un equipo, mediante la vigilancia de alarmas, el siguiente paso consiste en identificar qué módulo o pieza del equipo es la que está produciendo el fallo e interfiriendo en su correcto funcionamiento. Generalmente es necesario combinar mecanismos de test, diagnóstico y monitores de rendimiento para determinar la causa principal de un fallo. La rapidez y facilidad para detectarlo depende en gran medida de la calidad del diseño del equipo. Para facilitar la localización de fallos, es usual crear árboles de clasificación de modos de fallo con las rutinas de diagnosis asociadas.

Algunas de las funcionalidades asociadas a la localización de fallos son las siguientes:

- ***Diagnosis***: El mecanismo de diagnosis nos permite verificar si un equipo está trabajando con un rendimiento aceptable o deseado. La diagnosis no debe influir en el normal funcionamiento del equipo mientras se está realizando. Este tipo de herramientas también suele usarse para determinar si un equipo ha sido correctamente instalado cuando involucra a múltiples sistemas. El objetivo de la diagnosis es determinar qué equipo está fallando (localizarlo) para posteriormente ejecutar sobre él chequeos que permitan identificar la causa del fallo.
- ***Selección de pruebas***: Determina el diagnóstico que va a llevarse a cabo. Puede ser dirigido por un planificador o por un fallo detectado con un test. En el primer caso, la selección del módulo apropiado es relativamente simple dado que los módulos pueden ejecutarse individualmente. La estrategia consiste únicamente en determinar cuál ejecutar. En el caso de un fallo detectado con un test, el proceso de selección suele consistir en un conjunto de ficheros de comandos (*script*). La selección del *script* adecuado viene determinado por la naturaleza del fallo.
- ***Ejecución del diagnóstico***: La ejecución de un diagnóstico viene determinada en gran medida por el diseño del equipo. Deben considerarse aspectos como si se puede utilizar todos los recursos del sistema o si el equipo puede seguir trabajando mientras el diagnóstico se ejecuta en un segundo plano.
- ***Correlación de resultados***: Este proceso es mucho más costoso que el de correlación de alarmas. De todos los elementos que anuncian el fallo, el proceso de correlación debe discernir entre cual es el que ha fallado.

1.1.3 Chequeo

El concepto de chequeo es diferente al de diagnosis, dado que el primero involucra a múltiples sistemas mientras que el segundo suele ejecutarse sobre un único sistema o equipo. El proceso de chequeo consta de dos partes; (a) el *seleccionador,* que identifica que parte del circuito está fallando y (b) el *aislador* que, determina el módulo que contiene el fallo y debe ser sustituido.

Estos procesos se utilizan durante períodos de instalación para comprobar que el equipo ha sido instalado correctamente. Los criterios utilizados para determinar la bondad de los resultados suele ser más estrictos que en el caso de los tests del período de mantenimiento, más orientados a detectar problemas o verificar su reparación. Estos tests pueden ser *intrusivos* o *no intrusivos* según alteren o no el funcionamiento del sistema.

En el sistema suele encontrarse una base de datos conteniendo información acerca de la forma en la que ejecutar los diferentes tests, las condiciones que pueden aplicarse a los diferentes equipos y

sistemas y los resultados esperados. El acceso eficiente a esta base de datos constituye uno de los principales cuellos de botella de este tipo de chequeos.

Cuando se detecta un problema, una buena estrategia consiste en ir segmentando el sistema en subsistemas cada vez más concretos hasta determinar el segmento que está produciendo el fallo. De este modo podremos aislar más fácilmente el equipo o subred que ha producido el fallo.

Una incidencia puede producir múltiples fallos. Una buena política una vez localizado el fallo en cuestión consiste en examinar el conjunto de posibles fallos e inferir cuál constituye la causa principal en base a los resultados obtenidos.

1.1.4 Restauración del servicio

El mantenimiento del servicio en presencia de fallos implica el aprovisionamiento de una prudente cantidad de recursos de repuesto y la flexibilidad para reemplazar efectivamente esos recursos. Se pueden tomar varias estrategias de restauración del servicio:

- Aislar el equipo en el que se ha encontrado la causa principal del fallo, dejando a los demás equipos la carga de trabajo del equipo que funciona incorrectamente.
- Trasladar el servicio del equipo que falla a un equipo de repuesto. Esto puede llevarse a cabo si el diseño del sistema ha previsto esta posibilidad, dejando repuestos libres por cada grupo instalado u operativo.
- Utilizar un enrutamiento alternativo para separar el equipo del flujo de datos del sistema.
- Restaurar el software mediante una copia de seguridad, siempre que sea posible y no altere el rendimiento global del sistema.

La selección de una de estas estrategias debe realizarse en base a criterios de rendimiento y máxima disponibilidad del sistema resultante.

1.1.5 Reparación

El proceso de reparación depende del trabajo a realizar, repuestos, herramientas disponibles y, en gran medida, de la localización del fallo. El coste de reparación viene determinado por el asociado al trabajo y las piezas del recambio. Aspectos como un control de inventario, la elección de un experto para efectuar la reparación, etc., pueden ayudar en gran medida a reducir este coste.

Es importante también almacenar datos acerca de la reparación para mantener estadísticas del coste de reparación y la repercusión que el fallo ha tenido en la degradación de la calidad de servicio del sistema. Posteriormente se podrán usar estos datos para presupuestar el coste de mantenimiento de un sistema y las políticas de actuación.

1.2 Interfaces en gestión de fallos

En esta sección se hace hincapié en las características de las interfaces de gestión que son de utilidad en el área funcional de gestión de fallos. Una notificación viene caracterizada por el

protocolo, los atributos y comportamiento de los objetos gestionados, así como el de objetos adicionales definidos para tareas específicas de gestión de fallos.

En cuanto a los protocolos, los orientados a conexión tienen ventajas con respecto a los no orientados a conexión para la detección de alarmas, debido a la notificación del protocolo que se produce cuando falla una conexión. En otros casos, como el de la caída continua de conectividad, los orientados a conexión pueden dificultar la operación de gestión de fallos por la perdida de efectividad producida con los necesarios intentos de restablecimiento de conexiones. Para la emisión de eventos pueden utilizarse desde protocolos orientados a cadenas de caracteres (*strings*), *traps* SNMP [Bouloutas, 92] o los complejos protocolos basados en las especificaciones OSI [Rose, 90b] .

Con relación a los atributos y comportamiento que deben exhibir los objetos de gestión hay que tener en cuenta los siguientes aspectos:

- ***Eventos***: Los objetos gestionados deben implementar chequeos para responder a la ocurrencia de eventos. A la hora de seleccionar el chequeo más adecuado, el conocimiento implícito del objeto y del fallo hace más eficiente su selección.
- ***Estado de la alarma***: Debe existir un atributo que indique si no existe fallo, si el objeto esta en un estado transitorio de pasar de nivel de fallo a nivel operacional o si está en disposición de emitir alarmas con su correspondiente nivel de prioridad. Este atributo de un objeto puede ser utilizado para diseñar diagramas de estado y establecer situaciones de alarmas en base a la ocurrencia de ciertos cambios de estado.
- ***Utilización de atributos de estado***: En la gestión de fallos suele utilizarse los valores de atributos acerca del estado operacional, de uso y administrativo. Estos atributos indican la razón por la cual se ha pasado de un estado habilitado a uno deshabilitado, los filtros utilizados para emitir los eventos y el nivel de degradación que ha sufrido el sistema, entre otras cosas. El estado administrativo puede utilizarse también para bloquear un equipo mientras se esta llevando a cabo un test.

Existen también objetos adicionales que facilitan la gestión de fallos. Los objetos más importantes son los siguientes:

- ***Discriminador emisor de eventos (Event Forwarding Discriminator):*** Cada instancia de este objeto determina el destino de una alarma en concreto. Revisa todas las alarmas, les aplica el algoritmo de filtrado prefijado y envía una notificación al agente seleccionado. De este modo puede realizarse una clasificación de las alarmas y almacenando en lugares comunes de alarmas del mismo tipo.
- ***Log***: Este objeto contendrá el registro de alarmas que han ocurrido con vistas a nuevas referencias. Posteriormente pueden recuperarse aspectos como la secuencia temporal en que han ocurrido las alarmas.
- ***Test***: Un objeto de este tipo contendrá la rutina de test a desencadenar sobre un objeto de gestión o un grupo de objetos cuando sea invocada. Existen dos tipos de acciones; (a) no controladas, la acción invoca un test directamente y los resultados del test son recibidos como respuesta a esa acción y (b) controlados, la acción genera la creación de múltiples objetos test cuyos resultados son obtenidos mediante la interrogación de atributos de esos objetos test.

1.3 Gestión de fallos en Internet

La familia de protocolos de Internet está basado en un modelo centralizado con mensajes gestor-agente. Todas las primitivas de comunicaciones son iniciadas por el gestor, excepto la generación de alarmas (*traps*) que las inicia el agente de manera asíncrona. Una estación de gestión puede ser responsable de muchos agentes, cada uno de los cuales gestiona uno o varios dispositivos. No es práctico el uso de mecanismos de *polling* para consultar los posibles cambios que ocurren en la red, dado que existe el mecanismo de traps, con el que se consigue una mejora de procesamiento y de carga de la red. Es decir, la red no transporta información que el gestor no necesita, mientras que los agentes no responden a peticiones continuas irrelevantes por parte del gestor. La solución de compromiso consiste en utilizar los traps para advertir al gestor de un fallo, y que este se encargue de realizar *polling* para enterarse de la causa principal del mismo. Dentro de esta familia, el protocolo más importante es el *Simple Network Management Protocol* (SNMP).

El formato de la unidades de datos del protocolo es diferente al del resto de primitivas SNMP y la información que contiene es la siguiente:

- Tipo de PDU: En este caso indicará que se trata de una PDU de tipo *trap*.
- Iniciador: Identifica el subsistema de gestión que ha generado el *trap*. Su valor viene determinado por el identificador de objeto (OID) en el grupo *system*.
- Dirección IP del objeto que ha generado el *trap*.
- *Trap* genérico: Código que indica un trap predefinido genérico. Existen las siguientes posibilidades:
    - *coldStart* (0): Se produce una reinicialización de la entidad SNMP emisora que desencadena un error de inicialización. En principio, la causa puede ser un suceso accidental que genera un fallo grave.
    - *WarmStart* (1): Se produce una reinicialización de la entidad SNMP sin que ocurra ningún error.
    - *linkDown* (2): Señal de fallo en uno de los enlaces de los agentes de comunicaciones. El primer elemento del campo de la PDU de variables vinculadas indicará el nombre y valor de la instancia que refleja la interfaz referenciada.
    - *linkUp* (3): El enlace con el agente de comunicaciones se ha restablecido.
    - Fallo de autenticación (4): La entidad de protocolo ha recibido un mensaje de fallo de autenticación del emisor de la función requerida.
    - Pérdida de par EGP (5): Se ha perdido la conexión con la entidad EGP (*External Gateway Protocol*) vecina.
    - Trap específico (6): Se ha reconocido un trap específico que se indica en el campo correspondiente en la PDU e indica la naturaleza del trap.
    - Marca temporal: Tiempo entre la última reinicialización de la entidad que emite el trap y el momento de generación del mismo.
    - Variables-vinculadas: Información adicional relacionada con el *trap*. El significado de este campo depende de la implementación específica del *trap*.

El problema de SNMP [Bharat, 96] para la gestión de red es el localismo intrínseco, dado que sólo puede funcionar dentro de un entorno intranet y obtener información acerca de los dispositivos interconectados. Cuando necesitamos una gestión internet tenemos que recurrir a una de las extensiones más importantes de SNMP: el Remote Network Monitoring (RMON) [Thomas, 95].

Otro de los problemas de SNMP es la poca información que se transporta en cada trap, el operador de red dispone de muy poca información acerca de la prioridad de este fallo, la correlación con traps previos, información adicional acerca de la alarma, etc. El uso de sistemas gestores de bases de datos puede ayudar, aunque de manera estática, a sufragar estas deficiencias.

En ciertas ocasiones puede resultar interesante consultar gran cantidad de la información de gestión (*Management Information Base* - MIB) [Rose, 90b] para localizar la causa principal de un fallo. SNMP no es el protocolo más recomendado para este tipo de accesos masivos. Tampoco lo es la estructura de información de gestión utilizada, dado que no permite modelar complejas estructuras de datos para representar algunas situaciones de fallo. No es posible el diseño de interrelaciones entre objetos gestionados lo que reduce enormemente las posibilidades de correlación y operaciones basadas en el conocimiento previo acerca del sistema.

Por otro lado, el operador desea cierto grado de gestión implícita en el sistema, es decir, ante determinados eventos, se espera que el sistema genere acciones de respuesta a los mismos. En SNMP se pueden llevar a cabo acciones prefijadas mediante la consulta por parte del agente del valor de una variable que modifica el gestor cuando necesita la invocación de la acción. Sin embargo, no es posible el paso de parámetros en esta llamada, lo que da gran rigidez a este proceso.

1.4 Gestión de fallos en OSI

Dentro de la organización de los estándares de gestión OSI [Rose, 90] se encuentran los relativos a las funciones de gestión de sistemas. Estas funciones se agrupan por criterios de funcionalidad y no se corresponden con ninguna de las áreas funcionales de gestión (FCAPS) en concreto. Es decir, una función puede ser utilizada en más de un área funcional. En gestión de fallos, los estándares involucrados son los presentados en la tabla 1.

ISO/IEC	ITU-T	Título
101164-4	X.733	Alarm Reporting
101164-5	X.735	Event Report Management
101164-6	X.736	Log Control
101164-7	X.737	Security Alarm Reporting

Tabla 1. *Funciones de gestión de sistemas OSI involucradas en gestión de fallos*

El estándar 101164-5 [ISO 101164-5] establece los componentes para soportar la emisión remota de eventos y procesamiento local de eventos. El estándar se basa en el concepto de discriminador y encaminador de eventos (EFD). Estas entidades son responsables de filtrar eventos basándose en un determinado número de criterios de selección en base a los que se decide si se emite o no el evento. Del mismo modo, el discriminador establece los umbrales y criterios que deben ser satisfechos para que el evento sea encaminado.

El estándar 101164-6 [ISO 101164-6] define las operaciones de *log* dentro de un sistema de gestión. Especifica cómo debe almacenarse información acerca de los eventos y operaciones generados en objetos gestionados. Especifica los mecanismos de control temporal para cuando se produce el *logging*, cuando se cancela y cuando se reinicializa. Define también las operaciones para recuperar y borrar registros de *log*, así como la modificación de los criterios que se utilizan para crear estos registros.

El estándar 101164-7 [ISO 101164-7] define los procedimientos de seguridad basados en la recepción de notificaciones. El estándar especifica cinco tipos de alarmas de seguridad: Una violación de la *integridad* que se produce cuando se interrumpe el flujo de información entre dos entidades, es decir, se pretende borrar, insertar o modificar este flujo. Se produce una violación *operacional* cuando un servicio no puede ser prestado por un mal funcionamiento o una invocación incorrecta. Una violación *física* indica un problema en el acceso a un recurso. Una violación del *servicio de seguridad* indica que se ha producido un ataque a la seguridad del sistema. Una violación del tipo *tiempo restante* ocurre cuando un evento ha sido generado después del umbral de tiempo permitido.

OSI ha determinado los siguientes requerimientos funcionales para la gestión de fallos:

- *Detección y anuncio del fallo*: Establece el mecanismo de provisión de fallos al usuario, incluyendo la especificación de filtros que determinen esta notificación de alarmas. Define también los eventos que deben ser anunciados, cuando comenzar o parar una monitorización y los niveles de alarmas que debe notificar cada evento.
- *Diagnosis del fallo*: Activar tests predefinidos que permitan averiguar la localización del fallo y comprobar si los componentes hardware se encuentran en línea.
- *Corrección de fallos*: Cambios en valores de los atributos de un determinado recurso pueden realizar tareas de reinicialización. También se pueden iniciar peticiones de reconfiguración por parte de la red.

El protocolo *Common Management Information Service Element* - CMISE [ISO 9595] contiene una única primitiva para establecer el servicio de notificación de eventos. Esta primitiva es la M-EVENT-REPORT. Es la primitiva utilizada para notificar un evento a un gestor que lo ha solicitado. A diferencia de las otras primitivas, la notificación del evento es iniciada por el proceso agente. En OSI éste servicio puede ser confirmado o no confirmado. En la siguiente tabla se indican los parámetros de esta primitiva:

Parámetro	Indicación	Confirmación
Identificador de invocador	Obligatorio	Obligatorio
Modo	Obligatorio	No presente
Clase del objeto	Obligatorio	Opcional
Instancia del objeto	Obligatorio	Opcional
Tipo de evento	Opcional	No presente
Tiempo del evento	Opcional	No presente
Tiempo actual	No presente	Opcional
Evento de respuesta	No presente	Condicional
Errores	No presente	Condicional

Tabla 2. *Primitiva M-EVENT-REPORT*

Cada notificación se acompaña de un único identificador que inicializa el usuario y libera el usuario destino. El modo especifica si se trata de una primitiva confirmada o no confirmada. Los siguientes dos parámetros especifican la clase e instancia del objeto que ha generado el evento. A continuación se indica el tipo de evento, el tiempo en que ha ocurrido y parámetros con información que introduce el usuario acerca del evento que ha ocurrido.

Los errores que puede generar una primitiva de este tipo se indican en la siguiente tabla:

Error	Descripción
Invocación duplicada	El identificador de invocación ya ha sido utilizado en una notificación anterior.
Argumento inválido	El valor de información acerca del evento tiene un rango no válido.
Argumento no tipado	Uno de los parámetros utilizados no es reconocido como válido para utilizar en una asociación CMIS.
Atributo incorrecto	No se ha especificado el valor de un determinado atributo y no existe un valor por defecto para el mismo.
Clase incorrecta	La clase del objeto de gestión especificado no es correcta.
Instancia incorrecta	La instancia indicada no se corresponde con ninguna especificada.
Fallo de procesamiento	No se ha podido realizar el proceso asociado a la notificación indicada.
Limitación del recurso	La notificación no ha podido realizarse por limitaciones del recurso.
Operación no válida	La operación no es una de las que pueden ser usadas en una asociación CMIS.

Tabla 3. *Errores generados por M-EVENT-REPORT*

El conjunto de estándares ha sido publicado bajo el denominador común de funciones de gestión de servicios (*Service Management Functions* - SMF) [ISO 7498-4] . El objetivo de estas especificaciones es impedir la duplicación de estas funciones en las FCAPS. De este modo, una SMF puede soportar requerimientos de uno o más áreas funcionales. De las áreas indicadas cómo SMF la que más nos interesa en cuanto a gestión de alarmas es la de función de notificación de alarmas recogida en la norma X.733.

Event Handling.

El estándar 101164-4 [ISO 101164-4] define e identifica las categorías básicas de errores o tipos de notificaciones. También se modelan las distintas notificaciones de alarmas que puede notificar un sistema de gestión OSI. Estas notificaciones son asociadas principalmente a la gestión de fallos, proporcionando los tipos de eventos genéricos así como los tipos de errores que incluye, causas probables y medidas de priorización de fallos.

Las alarmas son tipos específicos de notificaciones definidas para identificar y detectar fallos y condiciones anormales. Se definen cinco categorías básicas de alarmas:

- ***Comunicaciones***: Usadas para indicar cuando un objeto detecta un error de comunicación. Usado principalmente con procesos que requieren transportar información de un punto a otro.
- ***Calidad del servicio***: Usado para un fallo o degradación de la calidad de servicio de un objeto.
- ***Procesamiento***: Indica un fallo de procesamiento en uno de los objetos.
- ***Equipamiento***: Indica un fallo en el equipamiento.
- ***Entorno***: Usado para indicar un problema en el entorno. Está principalmente asociado con una condición relativa al entorno de operación del equipo.

Las funciones de manejo de eventos del estándar OSI [Embry, 90] definen los mecanismos de control de encaminamiento de notificaciones. Se basa en la premisa de considerar que todas las notificaciones tienen potencialmente el riesgo de convertirse en eventos (M_EVENT_REPORT). Todas las notificaciones están sujetas al control de los *Event Forwarding Discriminators* (EFD) cuya tarea consiste en seleccionar notificaciones particulares y enviarlas a los distintos destinos indicados. Una notificación encaminada es conocida como un EVENT REPORT. El EFD es una clase de objeto gestionado con atributos especiales como el constructor discriminador y el destino que permiten establecer los criterios de selección de eventos para determinados destinos. Además, se indica un estado administrativo mediante el cual la operación de encaminamiento puede ser suspendida o cancelada. Otros parámetros opcionales permiten establecer destinos alternativos, orden de prioridad, destinos de respaldo (cuando no es posible alcanzar un destino prefijado), el modo de operación confirmado o no confirmado (que pueden utilizarse con un *timeout*). Estos objetos soportan encaminamiento selectivo y elección de destinos planificable temporalmente (semanal, diaria....). Los EFD también pueden emitir notificaciones acerca de cuando se ha creado, borrado o modificado EFD. Estas operaciones son gestionadas por EFDs específicos para este fin.

Cada una de estas notificaciones se define en el estándar X.721/ISO 10165-2. Todas las notificaciones incluyen los siguientes parámetros:

Parámetro	Descripción
Causa probable	Proporciona una calificación acerca de la causa más probable del fallo. Existen 57 tipos de causas probables identificadas en el estándar.
Problema específico	Proporciona información específica acerca de la causa del error.
Gravedad	Indicación del grado en el que el objeto se puede ver afectado. Existen 6 niveles: crítica, mayor, menor, aviso, indeterminada y asentida.
Estado de retroceso	Estado en el que se encuentra el objeto que está realizando una operación de retroceso.
Objeto de respaldo	Instancia objeto que proporciona los servicios de respaldo
Indicación de tendencia	Si está presente, indica que existen otras alarmas relacionadas que todavía no han sido asentidas y que la recibida es más, igual o menos severa que las anteriores.
Información de umbrales	Se incluye cuando la alarma se genera por el salto de un umbral. A su vez puede ser por un valor de un atributo, un nivel de un contador, un temporizador sobrepasado, etc.
Identificador de notificación	Identificador de la alarma que puede ser utilizado por alarmas correlacionadas.
Notificaciones correlacionadas	Conjunto de identificadores de notificaciones con los cuales se considera que la alarma está correlacionada.
Definición del cambio de estado	Incluye cuando se está en un estado de transición. Consiste en dos parámetros, el estado en que está y el tiempo en que ocurre el cambio de estado.
Atributos monitorizados	Identifica uno o más atributos del objeto y su valor en el momento en que ocurre la alarma.

Tabla 4. *Estructura de una notificación*

Parámetro	Descripción
Texto adicional	Texto adicional indicando la descripción del problema que puede estar ocurriendo.
Información adicional	Conjunto de información adicional que se quiere añadir a la alarma.

Tabla 4. *Estructura de una notificación*

La figura 1.2 muestra el modelo general de gestión-notificación de eventos que describe los componentes conceptuales para el envío remoto de eventos y procesamiento local de los mismos. Al menos un objeto de gestión envía notificaciones que son encaminadas mediante un emisor de eventos. Un detector de eventos y un módulo de procesamiento reciben la notificación generada localmente y construyen un evento con toda la información de la notificación enviada más información adicional como la marca temporal en que ha ocurrido, la clase, la instancia.... Este evento es enviado a todos los EFD que existen en el sistema local. En los EFD dispararan una acción y se determina que eventos serán encaminados a un destino particular. En los EFD es en donde se realiza verdaderamente el filtrado y planificación temporal que permite determinar el intervalo durante el cual el evento puede ser seleccionado para enviar. La gestión de eventos permite también modificar el estado y atributos de los EFD (dado que son objetos de gestión), de esta forma, un EFD pude ser inicializado, finalizado, suspendido o borrado. La planificación y opciones de filtrado también pueden ser consultadas y modificadas. Los EFD pueden también emitir notificaciones que son recibidas y procesadas por todos los demás EFD.

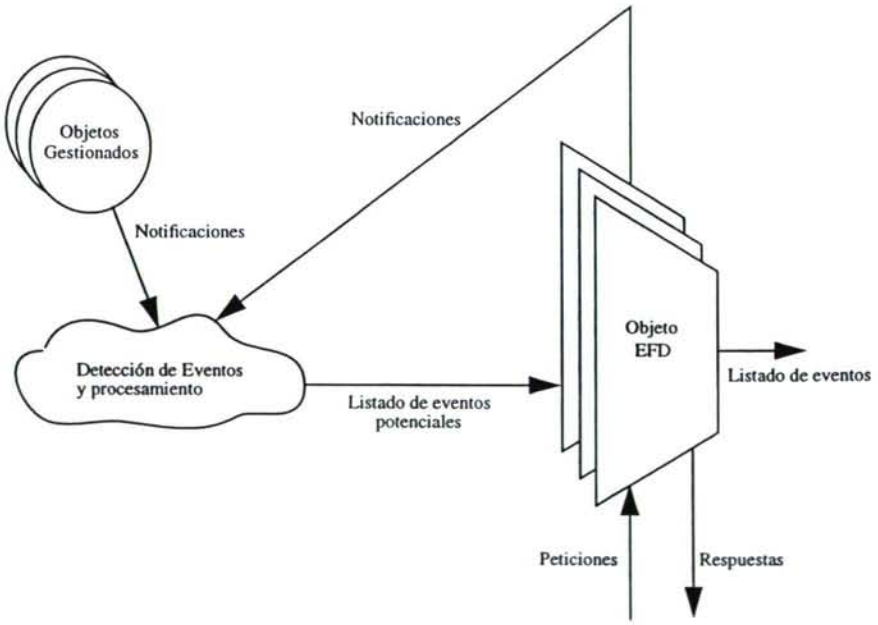

Figura 1.2. *Arquitectura general de un EFD*

El documento X.734/ ISO 10164-5 proporciona un modelo para el control y notificación de eventos. La función de gestión de eventos permite a un gestor controlar la transmisión de eventos desde el objeto gestionado independientemente de la definición de ese objeto. El log es un objeto gestionado que permite almacenar selectivamente los eventos bajo los criterios indicados por las funciones de control de log. Este objeto cuenta con un constructor discriminador que permite seleccionar los registros almacenados en el log, un estado y atributos que facilitan la planificación de los EFD. Dado que el log es un recurso finito, se contemplan dos posibilidades cuando se llena:

- Vaciarlo cuando se llene.
- Vaciarlo parcialmente cuando se llegue a su máximo.

Cada operación propuesta por un gestor debe pasar un control de acceso por parte del agente. Se indica al agente que objetos y operaciones sujetas a control de acceso para el gestor.

El correcto funcionamiento de la red se ve influenciado por eventos o notificaciones de situaciones anómalas. Cuando este evento modifica el normal funcionamiento de la red se le conoce como fallo y puede distorsionar el entorno de trabajo del usuario. Existen dos posibilidades de reconocer un evento y sus consecuencias; directamente, el usuario ve modificado su entorno de trabajo e informa al gestor de red; indirectamente, observación de cambios anormales en los valores de ciertas variables monitorizadas mediante una herramienta de gestión de red. Cuando ocurre un evento, en la mayoría de las situaciones, no puede determinarse la causa que lo ha provocado, pero sí el efecto, o en el mejor de los casos, el síntoma. La identificación de la causa depende, en gran medida, de la utilización correcta de monitores y estadísticas que esté utilizando el gestor de red.

Cuando se recibe una alarma de fallo, debe seguirle un test para localizar la causa principal y reconfigurar. Durante el proceso de reconfiguración, el dispositivo puede ser testeado para establecer que este proceso se realiza de acuerdo a la especificación. Esta asistencia ha sido estandarizada mediante las funciones de test de diagnóstico y confidencialidad con cinco categorías (conectividad, integridad de datos, integridad del protocolo, loop-back, echo y self-test).

Los objetos gestionados emiten notificaciones cuando ocurren ciertos eventos. Estas condiciones se especifican en un atributo del objeto gestionado. Se incluyen notificaciones en los estándares para creación, borrado, cambio de estado, cambio de atributos, emisión de alarmas de seguridad, etc. Puede contener atributos opcionales cuyo uso es importado de otras definiciones. Existen algunos problemas a la hora de definir notificaciones, los estándares no dejan claro el proceso de rellenado de alguno de los campos como eventos correlacionados. El mismo problema ocurre con la especificación de los contenidos de los logs y de los eventos potenciales que deben generarse.

1.5 Conclusiones

La gestión de fallos es una de las FCAPS con mayor relevancia en las modernas redes de telecomunicaciones debido a las interrelaciones que tiene con el resto de áreas. En las actuales redes de telecomunicaciones su correcta utilización es vital para garantizar por contrato un determinado nivel de QoS, asegurando que se detecta un fallo antes de que el usuario perciba su sus efectos significativos. Algunos de los problemas con lo que el operador se encuentra en una red son los fallos en los dispositivos, rendimientos ineficientes, ataques de seguridad y fallos en la contabilidad/utilización de los recursos. Los tradicionales sistemas de gestión presentan dos categorías de limitaciones, en base a criterios de información o de arquitectura, los cuales desarrollamos con mayor profundidad a continuación.

El principal problema de los tradicionales sistemas de gestión es que, si bien, alertan de la ocurrencia de un fallo, no proporcionan información adicional que permita identificar el origen del mismo. Muchos de los protocolos propietarios de gestión (y algunos de los estándares como SNMP) no han sido diseñados para transportar este tipo de información en sus PDUs. La familia de estándares OSI incorpora más información en las primitivas M-EVENT del protocolo CMIP, pero adolece de riqueza semántica asociada. Además, la cohabitación de alarmas de distintos orí-

genes (alarmas propietarias, traps SNMP y eventos CMIP) sé hace, en muchos casos, inviable, dado que el operador recibe información de las alarmas en diferentes formatos, con diferentes significados, dificultando enormemente la comprensión y tratamiento de las alarmas recibidas. ISO ha propuesto en su estándar X.733 el modelo de información a seguir para el intercambio de alarmas entre entidades de gestión. De este modo, se consiguen sistemas de gestión con un conjunto uniforme de tipos de notificaciones y con parámetros normalizados. La utilización de robustas bases de datos orientadas a objetos, simplifican y facilitan el acceso a la información de los eventos, permitiendo almacenar datos adicionales acerca de la ocurrencia de los mismos. Esta información es conocida implícita y previamente lo que facilita la localización del fallo.

Un segundo grupo de limitaciones de los actuales sistemas de gestión de fallos, proceden de su arquitectura centralizada, utilizada históricamente por razones de baja capacidad de los recursos de red y homogeneidad de los equipos. El enfoque centralizado de gestión separa lógica y físicamente las aplicaciones de los datos que maneja y los recursos de los procesos que los controlan. Los agentes actúan como simples almacenes recolectores y emisores de información a los gestores que en ciertos ratios se ven desbordados por los datos recibidos. En un entorno de gestión amplio y heterogéneo, la no escalabilidad de estas plataformas hace que el intercambio de datos entre agentes y gestor pueda exceder las capacidades de la propia red. Si el número de dispositivos, el número de variables a manejar y la velocidad de la red se incrementa considerablemente, el sistema se vuelve rápidamente inmanejable. Además, durante un período de fallo de la red, la política de enviar todas las notificaciones hacia un gestor central, no hace sino empeorar la situación, debido al retardo que estas comunicaciones introducen en la propia red.

Como se ha indicado anteriormente en este capítulo, la gestión de alarmas (objeto del presente trabajo) está englobada en la gestión de fallos. Concretamente realiza las funciones de vigilancia. Nos centraremos, a partir de ahora, en la mejora de aspectos relativos a la gestión de alarmas que van a permitir el diseño de sistemas de gestión de fallos más eficientes.

2 .- *Análisis y evolución de las arquitecturas de gestión*

Como base para afrontar el desarrollo de un modelo de gestión de alarmas, en este capítulo se analizan una serie de modelos o arquitecturas de gestión que se han considerado representativas dentro del amplio espectro de sistemas que pueden encontrarse en la literatura. Muchas de ellas no fueron diseñadas para la gestión de alarmas, pero su estudio nos permite determinar la bondad de la arquitectura para este tipo de gestión.

El capítulo comienza describiendo los criterios utilizados para analizar las diversas plataformas o sistemas de gestión elegidas. Posteriormente, se describen los modelos arquitecturales que tradicionalmente se han usado para gestión de red: centralizado, jerárquico, o distribuido. Dentro de cada apartado, se describen las características principales de este tipo de sistema y de relatan someramente los conceptos fundamentales de sistemas que representan cada una de los modelos expuestos.

Para finalizar el capítulo, en el apartado de conclusiones, se indican las características que debe tener un eficiente sistema de gestión de alarmas.

2.1 Medidas paramétricas

En este apartado se describe un conjunto de medidas paramétricas que permiten analizar, en que medida, alguna de las arquitecturas o modelos presentados a continuación son eficientes a la hora de construir un sistema de gestión de fallos. A la hora de afrontar el estudio de los diversos sistemas de gestión se han utilizado los siguientes criterios:

- ***Complejidad***: Se analiza la complejidad del sistema o de la plataforma desde dos puntos de vista: la dificultad de manejo de la misma por parte del usuario final y sus facilidades de configuración por parte del operador. La primera determinará el perfil del operador que la maneje. Muchos sistemas actuales de gestión resultan difíciles de manejar para los operadores, muy experimentados en el manejo de equipos pero con ciertas limita-

ciones con las últimas tecnologías de *GUIs* y programación de sistemas. La facilidad de configuración facilita al operador la personalización del entorno de trabajo, haciendo más amigable el entorno de operación por eso es un punto fundamental en el estudio de cualquier sistema informático.

- ***Flexibilidad de expansión***: Se determina en que grado el sistema permite aumentar la funcionalidad de gestión (tanto de los gestores como de los agentes), y lo complejo que puede resultar esta expansión. Además deben analizarse las posibles implicaciones que pueden tener estas mejoras. Si se trata de un sistema distribuido, debe analizarse en que medida esta distribución es transparente al operador, para que no necesite conocer de antemano ciertos datos del sistema, como localización de los agentes, del gestor, etc.
- ***Rendimiento***: Se determina como afecta la implantación de un determinado sistema de gestión al rendimiento global de la propia red. Se analizan implicaciones en la latencia, tiempos de respuesta, *throughput*, etc. así como la sobrecarga que se añade tanto a los dispositivos como a sus interfaces de red.
- ***Escalabilidad***: Se analiza como se comporta el sistema frente al incremento del numero de nodos en la red o el aumento de las prestaciones de los equipos existentes. Tiene implicaciones con el rendimiento aunque en muchos casos existen otras limitaciones que hacen a un sistema no escalable.
- ***Seguridad***: Muchos sistemas no utilizan sistemas seguros para el trasiego de información, resultando vulnerables en una red con posibles intrusos o ataques a su seguridad. En otros no es posible establecer mecanismos de autenticación, encriptación, control de acceso, etc. También se analizan las implicaciones de seguridad en las que pueden caer los sistemas que usan múltiples gestores especializados. Otro punto a tener en cuenta es la protección que el sistema aporta sobre los datos de gestión y el control que puede ejercerse sobre los recursos.
- ***Integrabilidad***: Actualmente las redes son heterogéneas y multiprotocolo. Los sistemas de gestión deben adaptarse a estos entornos para conseguir una gestión efectiva de la red. En muchos sistemas es posible la convivencia de información y protocolos diferentes mientras que en otros, si bien pueden admitir varios protocolos de gestión, mantienen una única base de datos de gestión con su exclusivo modelado de información (ver [Carneiro, 97b]). También resulta interesante que las los gestores o agentes puedan acceder a otras aplicaciones que ya utilice el operador: bases de datos, aplicaciones de gestión, agendas, etc. Esto es posible mediante *gateways* o con sistemas distribuidos basados en *CORBA* [OMG 91.12.1].
- ***Fiabilidad***: Se analiza como se comporta el sistema frente a fallos. Es decir como los identifica y como es capaz de restaurar el servicio que el sistema de gestión está ofreciendo al elemento gestionado. Se estudia también, la capacidad de autonomía de que pueden disponer los agentes cuando carecen de conexión con el gestor central.
- ***Información***: Se estudia tanto la capacidad de trasiego de información que pueda ayudar a encontrar el origen de un fallo como su capacidad de procesamiento. En muchos sistemas la información aportada sobre los recursos carece de semántica o es muy pobre. También se analizan las facilidades de almacenamiento de información y tratamiento de históricos, así como la distribución que presenta el almacenamiento de la información global de gestión.

- ***Estandarización***: Se analiza en que medida el sistema de gestión se adapta a los estándares actuales de gestión de red. Un sistema estándar será mucho más abierto y permitirá fácilmente su expansión y mantenimiento.

Muchos de los parámetros citados anteriormente no son aplicables en todos los sistemas de gestión estudiados. A continuación analizamos diversas categorías de sistemas, realizando una clasificación arquitectural previa donde se citan las características comunes que adoptan cada una de estas categorías de sistemas.

2.2 *Arquitecturas centralizadas*

El modelo de arquitectura centralizada es el mostrado en la figura 2.1. Consta de un proceso central (gestor) responsable de manejar globalmente la red, mediante la interacción con procesos remotos (agentes). Estos últimos son los responsables de recolectar información de los recursos a gestionar. El gestor controla la comunicación con los agentes, proporcionando toma de decisiones, control y registro de eventos de una manera centralizada. Esta aproximación cuenta con todas las ventajas y desventajas de una aproximación centralizada [Leinwand, 96], donde los principales inconvenientes residen en la difícil escalabilidad y degradación del rendimiento global de la red, cuando ésta se expande en tamaño o complejidad. Aún con todas las limitaciones que este tipo de sistemas, es el más utilizado actualmente, dado que es el adoptado por SNMP (protocolo de gestión más utilizado actualmente).

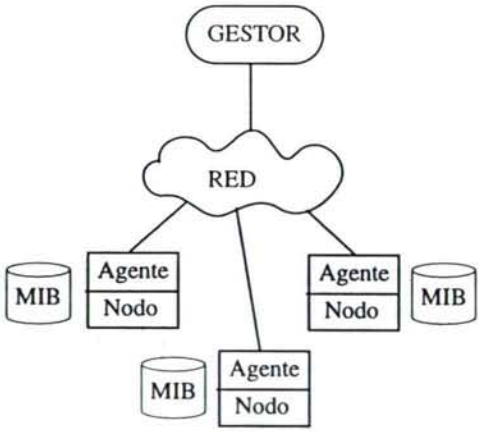

Figura 2.1. *Entorno centralizado de gestión.*

En la figura 2.2 pueden apreciarse los principales problemas asociados a un paradigma de gestión centralizado. Muchos de ellos están íntimamente relacionados con las comunicaciones extremo a extremo entre gestores y agentes. Concretamente podemos resumirlos del siguiente modo:

- ***Problema de inaccesibilidad***: Un fallo en un equipo de comunicaciones puede hacer inaccesible al agente desde el gestor. Durante este período, el gestor no puede realizar cálculos sobre información de ese elemento de red. Además, en el caso de que el o los agentes estén al otro lado de un frontal de comunicaciones, la subred quedará sin gestión hasta que se restablezca la comunicación. Esta situación puede solventarse mediante la instalación de otro gestor más allá del equipo de comunicaciones o bien, almacenando en históricos todos los datos que ocurren durante la ruptura de la comunicación.

- ***Ancho de banda***: Un gestor puede utilizar una línea de baja velocidad, como pe. una línea *WAN*, para gestionar parte de la red. La baja velocidad de esta línea puede limitar las operaciones de gestión, resultando inoportunas en casos de congestión de la misma. Puede resolverse instalando un gestor intermedio. El gestor enviará información de alto nivel pre-procesada al gestor intermedio, reduciendo de este modo el tráfico de gestión. Esto es la base de las arquitecturas conocidas como *midlevel managers* [Case, 93b].

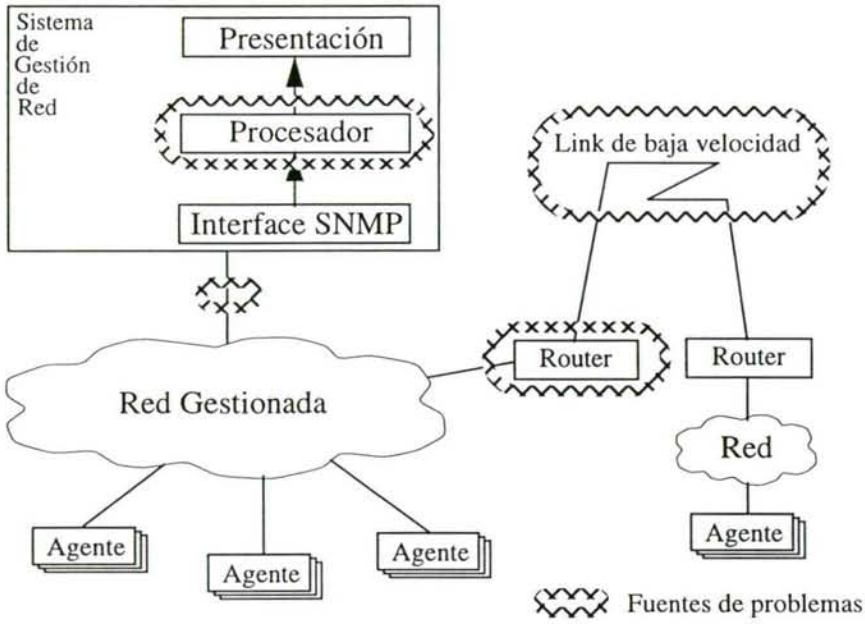

Figura 2.2. *Problemas de la gestión centralizada*

- ***Número de agentes***: El dispositivo que actúa de gestor puede incrementar sustancialmente su carga de *CPU* debido al incremento de agentes a gestionar. La escalabilidad de este tipo de sistemas puede representar un grave problema. Cuando la red crece rápidamente, el sistema resulta caro y complejo de escalar. Además, la necesidad de interrogar periódicamente a todos los gestores desde una única localización, puede sobrepasar la capacidad de procesamiento del dispositivo en el que se encuentra la plataforma de gestión. Una solución a este problema consiste en agrupar los agentes por áreas y asignar a cada grupo un gestor, posteriormente todos los gestores de área se intercambiarán información global acerca de la red [Case, 93].
- ***Capacidad de las interfaces***: Si el trasiego de información entre los agentes y el gestor es muy elevado, la interfaz del gestor que debe capturar toda la información a procesar, constituye un potencial cuello de botella. La solución a este problema consiste en separar el alto número de peticiones a procesar, mediante su distribución en varias máquinas. Sin embargo, con esta alternativa surgen los problemas de ubicación de estos equipos y de la necesaria coordinación entre los mismos. En una estructura jerárquica, el gestor de gestores (*MOM*) realiza estas funciones de coordinación. Otra posibilidad consiste en establecer comunicaciones entre gestores al mismo nivel, sin embargo, esto incrementa la complejidad del sistema de gestión, estos sistemas son conocidos como redes de gestores.

- ***Tolerancia a fallos***: Aunque, como se ha citado anteriormente, podemos mantener réplicas de la información de gestión, no todas las funciones de gestión dependen de la estación central. El hecho de mantener respaldos totales del sistema no resulta plenamente seguro por lo que el sistema no es redundante a fallos.
- ***Autonomía de los agentes:*** Los agentes se comportan como meros recolectores y emisores de información hacia la estación central. Su capacidad de procesamiento en periodos de fallo es muy limitada y su baja o nula capacidad de tratamiento de la información, no hace sino incrementar la congestión de la red.

Sin embargo, el modelo centralizado también tiene algunas ventajas inherentes [Lillian, 89]. La plataforma de gestión se encuentra en una única máquina de la red, lo que facilita determinadas funciones de gestión, como aquellas que se realizan sobre el conjunto de la información recibida, pe. correlación de alarmas. Dado que el sistema utiliza una única base de datos, es sencillo llevar a cabo funciones de respaldo. El sistema puede ser replicado en otra computadora a intervalos regulares, incorporando al sistema un cierto grado de tolerancia a fallos. La centralización de la información, facilita la interpretación y correlación de los datos, potenciando la protección de los mismos frente a accesos no deseados. La seguridad es más fácil de mantener ya que se puede restringir el acceso a esta información con mecanismo de bajo nivel (S.O.) de la máquina que esta ejecutando la plataforma de gestión. Además, aunque se trata de un sistema centralizado, es posible disponer de varias consolas de operador, en distintas máquinas de la red que se comuniquen con la plataforma de gestión mediante algún sencillo protocolo de transporte. Esto facilita el encaminamiento de eventos hacia diversas consolas de operación en las que reciben tratamiento personalizado.

2.2.1 Simple Network Management Protocol (SNMP).

El origen y evolución de este protocolo están ligados al de *TCP/IP*. En Noviembre de 1987 se presentó el *Simple Gateway Monitoring Protocol* (*SGMP*) que constituyó el primer protocolo de gestión. La necesidad de utilidades de gestión de propósito general, hace que surjan nuevas aproximaciones como el *Simple Network Management Protocol* (*SNMP*), el *High-Level Entity Management System* (*HEMS*) y el *CMIP Over TCP/IP* (*CMOT*), estos dos últimos muy poco utilizados en la actualidad.

En su origen, SNMP era una especificación interina, y como tal fue diseñado e implementado; tenía que servir para comunicar con dispositivos de red mientras que el estándar OSI se terminaba de materializar hacia el final de la década de los ochenta. Se suponía que SNMP desaparecería en cuanto OSI fuese una realidad, pero no ocurrió así. En 1993, cuando OSI terminó de madurar, SNMP ya tenía tres años de vida y estaba implementado en cientos de productos. Lejos de ser una solución provisional, SNMP se ha convertido actualmente en el estándar *"de facto"* de la gestión de red. Su popularidad se debe a que, como proclaman sus siglas, el protocolo SNMP es sencillo de implementar en *hardware*. Tras sacudirse la etiqueta de *"protocolo de transición"*, SNMP está evolucionando en su propia línea. Actualmente, SNMP es el protocolo de gestión más empleado en redes de datos. Originalmente, se diseñó para gestionar redes TCP/IP y *Ethernet*. Más tarde se implementaron nuevas versiones independientes de TCP/IP (SNMP sólo requiere mecanismos de transporte basados en *datagramas* para operar).

SNMP incluye dos componentes principales: (1) una arquitectura para la organización de la información de gestión en *Management Information Bases* (MIB) y (2) un protocolo de consultas a estas bases de información.

SNMP se basa en un modelo gestor/agente. Un agente SNMP está formado por *software* capaz de responder a consultas realizadas por un gestor ó estación SNMP acerca de la información definida en una base de información de gestión (MIB). Por lo tanto, cualquier dispositivo que proporcione información sobre la MIB a la estación debe tener obligatoriamente un agente SNMP. Este modelo se ilustra en la figura 2.3.

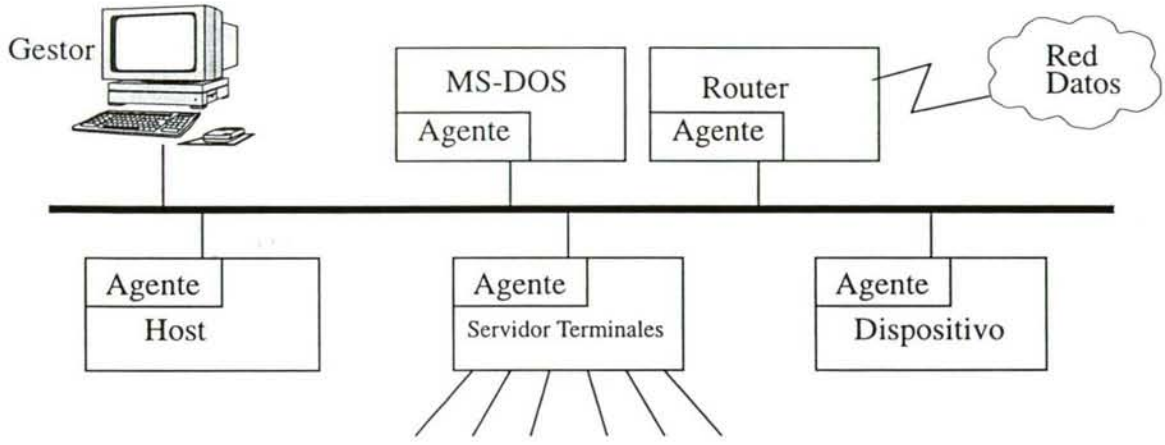

Figura 2.3. *Modelo Gestor/Agente SNMP*

Las MIBs, por tanto, proporcionan un modelo de organización de datos jerárquico para la información de gestión (MIB). SNMP utiliza el árbol de registros ISO como directorio de la información de gestión. La figura 2.4 muestra la estructura de este árbol, donde los nodos son etiquetados con un nombre (identificador global) y un número (identificador relativo). De este modo, cada nodo es identificado por la concatenación de los números desde él al directorio *root*. El subárbol *internet* es identificado por el camino 1.3.6.1 y es utilizado por esta organización para registrar sus estándares. Consta de tres subárboles asociados con la gestión: *management*, *experimental* y *private*. En el primero se registran los estándares aprobados de la organización (pe. MIB-II), en el segundo, aquellos estándares que se encuentran en estudio (pe. RMON2) y en la rama *private*, las MIBs que aportan diversos fabricantes de productos.

Hay que destacar que esta estructuración de la información de gestión es estática. La localización de la información en el árbol MIB se determina en el momento del diseño del mismo. Una vez finalizado el diseño de la MIB, las únicas modificaciones que se pueden realizar son: el cambio de los valores almacenados en la base de datos y la creación y borrado de instancias dentro de una tabla de la MIB. Por otro lado, la definición de la estructura de una MIB proporciona únicamente información sintáctica. Dos variables iguales pueden ser interpretadas de diferente modo, no se incorpora capacidad semántica acerca de los datos almacenados.

El contenido de las MIB está basado en la estructura de la información de gestión (SMI) definida en los términos de un lenguaje de descripción de la sintaxis de estructuras de datos. Identifica los tipos de datos que pueden ser usados en una MIB, como se representan y nombran los recursos en

la misma, y como se codifican. La filosofía de SMI consiste en simplificar al máximo la estructura de las MIB, solo pueden almacenarse tipos de datos simples: escalares o cadenas bidimensionales de escalares. La SMI introduce seis tipos de datos básicos, además de los universales de ASN.1 [ASN, 90]. Esta limitación en los tipos y tamaño de los datos simplifica en gran medida los procesos de codificación/decodificación de la información almacenada, así como su organización, protocolos de acceso a la información de gestión e implementación de monitores. SMI también introduce una macro de ASN.1, denominada *OBJECT-TYPE* que sirve de ayuda en la definición de nuevos objetos en los nodos del subárbol. Esta macro aporta facilidades para definir variables de gestión y asociarlas con tipos de datos, modos de acceso (lectura, escritura o lectura/escritura), el estado (obligatorio, opcional) y la localización estática dentro del árbol.

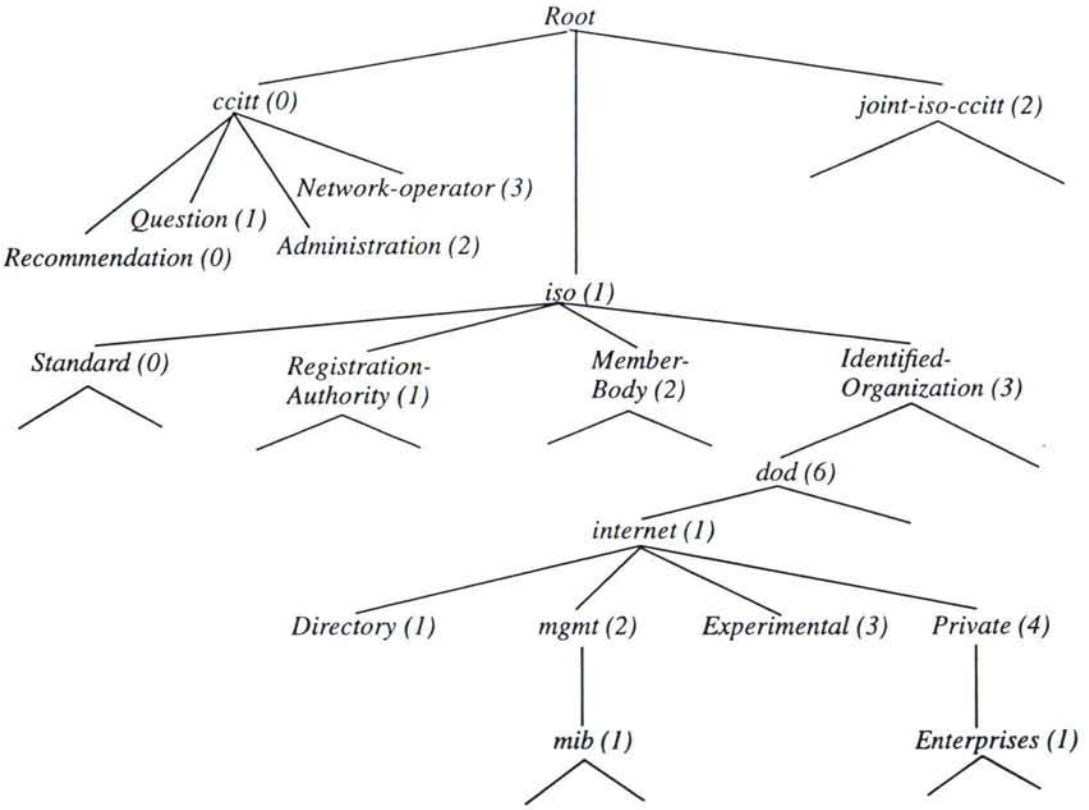

Figura 2.4. *Árbol de registros ISO.*

Esta estructuración de la información tiene, sin embargo, algunas limitaciones:

- ***Rigidez***: El contenido de la MIB debe ser indicado en tiempo de diseño. Posteriormente puede ser necesaria información que no reside en la MIB, por parte de operadores o aplicaciones. Por tanto, sumarizaciones que no se encuentren en la MIB deben ser llevadas a cabo por el gestor lo que traduce un mayor tráfico de gestión por la red.
- ***Ausencia de acciones directas***: Un agente debe realizar un comando SET sobre una variable predefinida para invocar un procedimiento. Esto puede ser válido en acciones sin paso de parámetros, las demás acciones requieren un SET previo con el valor de estos parámetros. Esta manera de actuar lleva aparejado el necesario mecanismo de sincronización agente-gestor que es muy difícil de implementar en un entorno de este tipo.

- ***Interrelaciones***: Con esta estructura no se incluyen relaciones complejas entre objetos de gestión. De este modo cuando se requiere información que involucra a varios objetos de la MIB con interrelación, es necesario recuperar datos de las tablas originales y las de las que implementan esta relación.
- ***Recuperación global***: Cuando es necesario recuperar el contenido de toda una tabla, es necesario realizar sondeos fila a fila. Esto incrementa la complejidad del protocolo y aumenta la información de gestión que circula por la red. Este aspecto ha sido mejorado con la versión 2 del protocolo SNMP.
- ***Filtrado de datos***: Un gestor puede estar interesado únicamente en datos de una determinada variable que verifiquen una condición para evitar la transmisión de gran cantidad de datos innecesarios.

Además de las MIBs, el otro componente significativo de SNMP es el protocolo de comunicaciones (la unidad de protocolo de datos o PDU, *Protocol Data Unit*) que emplea el software de gestión para comunicarse con los agentes. Estos mensajes viajan dentro de la estructura de un UDP (*User Datagram Protocol*). Existen los siguientes tipos de primitivas:

- ***GetRequest***: Obtención del valor de la variable de la MIB de un dispositivo de red.
- ***GetNextRequest***: Obtención del valor siguiente (en orden lexicográfico) al solicitado en la anterior primitiva *GetRequest*.
- ***GetRespond***: Empleada por el agente SNMP para devolver al gestor los datos solicitados.
- ***SetRequest***: Primitiva para la modificación de variables en la MIB del agente SNMP.
- ***Trap***: Esta es una primitiva especial que los agentes pueden enviar asíncronamente a un *manager* para notificar determinadas condiciones o estados, previamente definidos, que se dan en el dispositivo gestionado.

Con el objetivo de integrar agentes propietarios con la gestión SNMP, se definieron los *agentes Proxy* SNMP. Estos agentes permiten a un gestor SNMP monitorizar y controlar elementos de red que no dialogan vía el protocolo SNMP, sino a través de un protocolo propietario, tal como muestra la figura 2.5. El agente SNMP *proxy* actúa como un conversor de protocolos para, en esencia, traducir los comandos SNMP al esquema propietario. Esta estrategia facilita la integración de entornos propietarios en el entorno abierto y estándar de SNMP.

Figura 2.5. *Concepto de agente proxy en SNMP.*

Otro aspecto importante lo constituye la característica de que SNMP es un protocolo no orientado a conexión. Primitivas como GET, GET-NEXT y SET son confirmadas por la correspondiente GET-RESPONSE. La plataforma de gestión detecta si se ha perdido una petición al no recibir el correspondiente retorno. Sin embargo, los *traps* (alarmas enviadas de forma asíncrona) son gene-

rados por el agente y no son confirmados, existen métodos para confirmar estos mensajes (variable con número de repeticiones o variable de confirmación) pero ante fallos en la red o sobrecarga, ninguno de los métodos es bueno.

Con la especificación SNMP original, no se puede comprobar el origen de un mensaje de gestión, ni la intercepción de comunicaciones. Sin la capacidad de comprobación o autenticación, SNMP es vulnerable a ataques para modificar a desactivar configuraciones de red. En consecuencia, muchos fabricantes de equipos SNMP optan por no implementar el mandato SET de SNMP. Ello reduce las capacidades de los productos de estos fabricantes a únicamente monitorización.

Los primeros intentos de estandarización para resolver los problemas de SNMP citados con anterioridad se plasmaron en los protocolos: *Secure* SNMP y SMP (*Simple Management Protocol*). Esto planteó serios problemas dado que, por ejemplo, *Secure* SNMP no era compatible con SNMP. Existían diferencias en el formato de la cabecera del mensaje. Sin embargo, una vez analizadas las ventajas de estos protocolos, en 1992, los miembros de la comunidad de Internet decidieron desarrollar un nuevo protocolo con las principales características de los anteriores. De este modo surgió SNMPv2. Las principales incorporación de la segunda versión de SNMP son la incorporación de mecanismos de seguridad que preserven la integridad, autorización y autenticación de los mensajes de gestión y la posibilidad de distribución de las tareas de gestión gracias a la comunicación *manager-manager.*

El concepto clave de SNMPv2 es el establecimiento de entidades *Party* [Sloman, 94] . Esta entidad permite establecer diferentes políticas de acceso y seguridad dependiendo de la combinación de agentes y gestores que accede a la información. Para establecer estos mecanismos utiliza una MIB especializada, denominada *Party MIB*. En las PDUs de SNMPv2 se incorpora información acerca de la *Party* origen y *Party* destino, el contexto de operación (vista de la MIB a la que puede acceder esa entidad) e información de autorización. El campo destino es encriptado para garantizar la privacidad del mensaje.

En cuanto a los mecanismos de seguridad que se establecen en SNMPv2, cada PDU SNMPv2 incorpora además la siguiente información:

- ***Privacidad***: Protocolo que se está utilizando para garantizar que nadie puede leer la información que circula entre el agente y el gestor. Se utiliza el *Data Encription Standard* (DES), con sus claves públicas y privadas.
- ***Autenticación***: Protocolo utilizado para garantizar que una tercera entidad no ha modificado la información de gestión. Se utiliza el *Message Digest 5* (MD5), con sus claves públicas y privadas, así como información temporal del mensaje (marca de creación y tiempo de vida).
- ***Control de acceso***: Definición de la vista de la MIB que se está solicitando, con el objeto base e información de inclusión/exclusión del subárbol correspondiente.

Una de las MIB desarrolladas junto con la especificación de SNMPv2 es la M2M MIB que proporciona el soporte para la gestión distribuida de gestores, facilitando la comunicación gestor-gestor a través de una nueva primitiva denominada *INFORM REQUEST.* Dentro de esta MIB se define también un grupo de alarmas y la información asociada que se introduce en una primitiva *INFORM REQUEST* cuando un gestor intermedio quiera notificar algo a un gestor de nivel superior.

Existen otras diferencias con SNMPv1 que pueden resumirse del siguiente modo:

- Se incorpora una nueva PDU llamada *GetBulkRequest*, que reduce el número de peticiones y respuestas mejorando así el rendimiento al recuperar árboles MIB enteros. De esta manera se optimiza la recuperación de grandes cantidades de información de gestión.
- Se ha optimizado la PDU *GetRequest*, de manera que la PDU *GetResponse* asociada devuelva una lista de variables aunque algunas de ellas no puedan ser obtenidas, bien por identificación errónea, bien por permisos insuficientes por parte del gestor. Esta optimización se ha aplicado también a *GetNextRequest*.
- Soporte multiprotocolo estandarizado que, al contrario que SNMPv1, además de funcionar sobre redes IP, también se ha estandarizado para redes como *Appletalk, Novell IPX* y *CLNS* (*OSI Connectionless Network Service*). Debido a que todos los protocolos, incluido IP, se encuentran en el ámbito de red en el modelo OSI, mientras que SNMP es un protocolo a nivel de aplicación, es posible cambiar el protocolo de red sin que ello afecte para nada a SNMPv2.
- Compatibilidad con SNMPv1 Debido a que la SMI de SNMPv2 es un superconjunto de la SMI de SNMPv1, todas las definiciones de la MIB de SNMPv1 son perfectamente compatibles con la MIB de SNMPv2. En lo que respecta a las nuevas PDUs, *GetBulkRequest*, por ejemplo, puede implementarse como una serie de primitivas *GetNextRequest*. La coexistencia puede implementarse a través de un agente *proxy* que realice la traducción o mediante la implementación de un gestor que maneje indistintamente SNMPv1 y SNMPv2. Para comunicarse con un agente hace uso de la información almacenada en una base de datos local con información de los protocolos soportados por cada agente. Esta última solución es la más extendida.

Con todas estas mejoras, SNMPv2 ha ganado en seguridad y descentralización, pero sigue adoleciendo de los problemas derivados de la no adopción del paradigma de orientación a objetos:

- ***Encapsulación***: SNMP no proporciona un mecanismo formal para la encapsulación de variables, lo que representa un serio contratiempo en el mantenimiento de la consistencia.
- ***Reusabilidad***: El registro de una variable en la MIB depende de como ha sido añadida en el árbol. Por tanto, una definición de un tipo de variable no puede ser utilizada para ser usada en otra parte diferente de la MIB.
- ***Herencia***: SNMP no soporta esta característica, lo que impide que en la definición de variables se utilicen otras para su especialización.
- El sistema de nombrado de variables de SNMP conlleva el cambio de la aplicación de gestión cada vez que una variable cambia de lugar en el MIB.
- Limitación en el tipo de datos que pueden utilizarse para el diseño de la MIB. En principio sólo los de ASN.1, ya que aunque se pueden formar más, complican enormemente el diseño de los agentes al ser necesario especificar para cada uno las reglas de codificación/decodificación (BER) necesarias.

Las limitaciones no deben entenderse como críticas que busquen la desaparición de SNMP, sino más bien como la identificación de áreas en las que se debe hacer un mayor esfuerzo para buscar

soluciones, bien mediante nuevas versiones, o bien implantando nuevos modelos o paradigmas de gestión.

2.2.2 Remote Monitoring Protocol (RMON).

La revisión más importante, junto con SNMPv2, que se ha llevado a cabo sobre SNMP es la MIB RMON. Se añade nueva funcionalidad sin modificar el protocolo, por lo que es compatible con SNMP. Con este último protocolo, podemos obtener estadísticas acerca de cualquier elemento de la red gestionada, entrada/salida de cualquier equipo, pero no de la red en global. Para esto se utilizan monitores que actúan en modo "*promiscuo*" escuchando los paquetes que circulan por la red. Estos monitores actúan de forma autónoma, cuando es necesario la comunicación con un gestor central, se requiere la utilización de monitores remotos.

Durante la fase de diseño de RMON, en la que se definía principalmente una MIB y sus funciones e interfaces para comunicación entre consolas de gestión basadas en SNMP se establecieron los siguientes fines u objetivos:

- *Operación Off-Line*: Usualmente, puede producirse un corte en la comunicación entre el gestor y el monitor remoto, en estos casos, el monitor remoto debe ser capaz de actuar autónomamente y realizar procesos de sumarización para entregar resultados al gestor en cuanto se restablezca la comunicación. Esta cualidad de los agentes RMON puede ser aprovechada también para reducir el número de interrogaciones o *polling* que el gestor tiene que realizar, lo que redunda en un mejor rendimiento de la red gestionada.
- *Monitorización proactiva*: El monitor puede llevar a cabo operaciones de test y diagnóstico por iniciativa propia, descargando al gestor de estas operaciones. Posteriormente informará al gestor de los resultados de las pruebas realizadas.
- *Detección y notificación de problemas*: Dado que el monitor está continuamente realizando operaciones de vigilancia del tráfico que circula por la red, puede ser programado para detectar algunos problemas de performancia que surjan, notificando estas ocurrencias al gestor(es) central(es).
- *Datos de valor añadido*: El monitor remoto puede llevar a cabo análisis específicos de los datos recolectados en su subred, relegando al gestor de esta tarea. Pueden realizarse análisis específicos de la subred que no serían viables a través del gestor, dado que éste no se encuentra conectado directamente a esta subred.
- *Múltiples gestores*: En un entorno de *internetworking* pueden existir más de una estación de gestión, por razones de rendimiento, funcionalidad o capacidades/necesidades de gestión de una organización. El monitor remoto puede ser configurado para atender a cualquier estación de gestión de manera concurrente.

La base del desarrollo del estándar RMON ha sido la definición de la RMON MIB que añade más funcionalidad a la MIB-II definida para SNMP, debido principalmente a:

- *Configuración*: Un agente puede ser configurado para realizar un preprocesado de la información que va a coleccionar. Para implementar esta funcionalidad, para cada grupo de la MIB se define una tabla de control y una o varias tablas de datos. Apuntes en la tabla de control explicitan la funcionalidad que se pretende para las filas de las tablas de datos cuyo índice de control coincida con el de la tabla de control. La figura 2.6 muestra un ejemplo de configuración con este tipo de organización. En la tabla de control existen los campos: *Con-*

trolIndex que identifica unívocamente a cualquier fila de la tabla, *ControlParameter* es uno de los parámetros aplicables a todas las filas de datos controladas por esta fila de control (en este ejemplo, sólo uno), el *ControlOwner* indica el propietario de esta fila y el *ControlStatus* indica el estado en que se encuentra. La tabla de datos es indexada por el *DataControlIndex* y el *DataIndex*, el primero indica que fila de la tabla de control gestiona esa fila de datos y el segundo identifica unívocamente a las filas de datos que son controladas por una misma fila de la tabla de control.

- *Invocación de acciones*: SNMP no proporciona un mecanismo para la invocación directa de comandos, sin embargo, es posible utilizar la operación *SET* para invocar acciones. En la RMON MIB, se utiliza un objeto que representa los estados por los que puede transcurrir la ejecución de una acción que se inicia con el cambio de valor del estado de ese objeto.

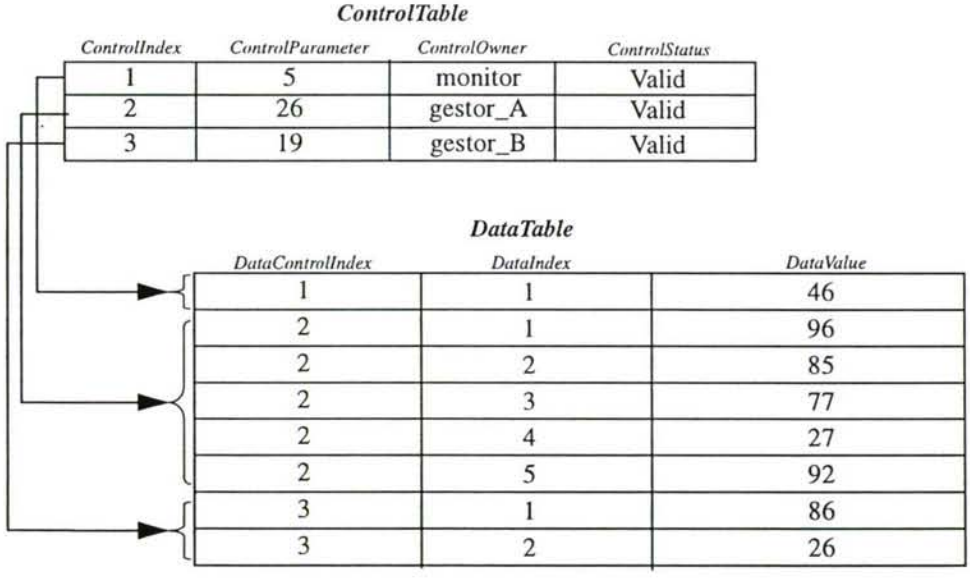

Figura 2.6. *Ejemplo de configuración con tablas de control y de datos.*

Cómo se indica anteriormente, uno de los objetivos de la monitorización remota es la posibilidad de soportar acceso por medio de varios gestores. Sin embargo, esta configuración puede desembocar en los siguientes conflictos:

- La concurrencia de accesos por parte de los gestores puede provocar un exceso de la capacidad del monitor.
- Un gestor puede provocar el bloqueo de algún recurso durante un largo periodo de tiempo.
- Un error o fallo en un gestor implica el bloqueo indefinido de los recursos apropiados por el mismo.

Para solucionar los problemas anteriores se ha colocado un identificador de propietario a cada una de las filas de una tabla de control. De este modo, una vez conocido el propietario de los recursos apropiados puede negociarse con ellos la liberación de los mismos. Es conveniente resaltar que esta etiqueta no actúa como un control de acceso, cualquiera de los gestores con permisos en esa *comunidad* puede llevar a cabo acciones sobre las tablas, está fuera de la especificación de RMON el control de las mismas. Además, es posible que otros gestores compartan una funcionalidad expresada en una fila de la tabla de control por un gestor concreto, sin embargo, sólo éste tiene permisos para modificar la fila.

Para la gestión de tablas se añade claridad a los mecanismos de SNMP mediante la inclusión de dos nuevos tipos de datos en el SMI. El *OwnerString* identifica al propietario de la fila, que es el que está autorizado a realizar operaciones sobre la misma. El *EntryStatus* indica el estado en que se encuentra la operación que se está llevando a cabo sobre la misma. De este modo, como se indica en la figura 2.7, cuando un gestor quiere crear una nueva fila el estado se coloca a *CreateRequest*, en el momento en que el monitor atiende la petición el estado pasa a ser *UnderCreation*. Una vez que se termina satisfactoriamente su estado para a ser *Valid*. Cualquier petición de creación de la misma fila, por parte de otro gestor, mientras la fila está *UnderCreation*, genera un error que es indicado al segundo gestor. El borrado o modificación de los parámetros de una fila hace que su estado pase a ser *Invalid*.

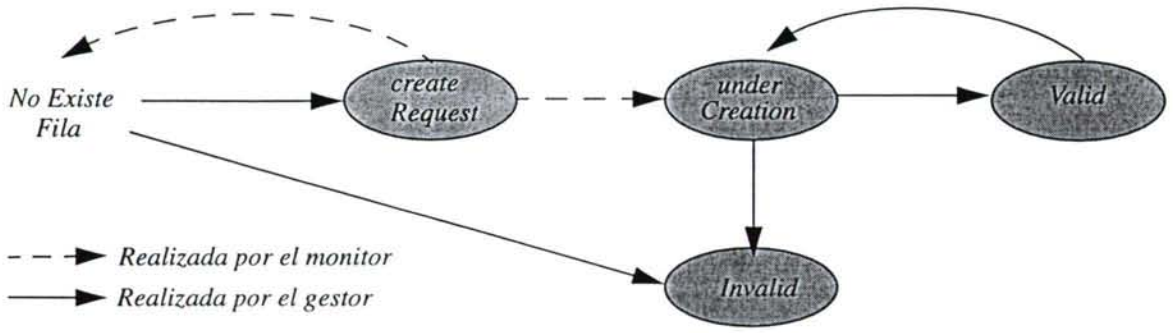

Figura 2.7. *Posibles transiciones del estado de EntryStatus.*

Una de las extensiones de RMON consiste en monitorizar el tráfico por encima de la subcapa MAC, los estudios referentes a llevar a cabo estas tareas se han concretado en la especificación de RMON2, que está basado en la decodificación de paquetes de las capas de red a aplicación del modelo OSI, lo que facilita que un agente RMON pueda observar paquetes pertenecientes a subredes vecinas en el caso de un *router* por medio o almacenar estadísticas basándose en la utilización de una determinada aplicación. RMON2 introduce dos nuevas funcionalidades con respecto a la versión primitiva en cuanto a la indexación de tablas: el uso de objetos índice que no forman parte de la tabla que indexan y el uso de filtros de indexación basados en el tiempo.

2.2.3 Common Management Information Protocol (CMIP).

La figura 2.8 muestra la arquitectura general de un modelo de gestión OSI. Se distingue entre objetos gestionados (MO) [ISO 10165-2] (cualquier vista de gestión OSI de un recurso) e Información de gestión (MI) [ISO 10165-4] (información asociada con un MO que es actualizada por un protocolo de gestión OSI que controla y monitoriza el objeto). Los MO residen en varias capas del modelo OSI, mientras que la MI reside en el *Management Information Tree* (MIT). Esta base de información contiene los datos asociados a los MO. Los gestores de capas son los responsables de mantener la asociación entre la información de la MIT y los MO, que constituyen, por tanto, abstracciones de los recursos y de sus propiedades. Para realizar estas abstracciones se usan las técnicas de orientación a objetos. De este modo un recurso es asociado con un vista de gestión en términos de atributos, que describen las propiedades de los recursos, operaciones, que pueden lle-

varse a cabo sobre el objeto, comportamiento, que indica como responde el objeto a las operaciones y notificaciones que puede emitir ese objeto.

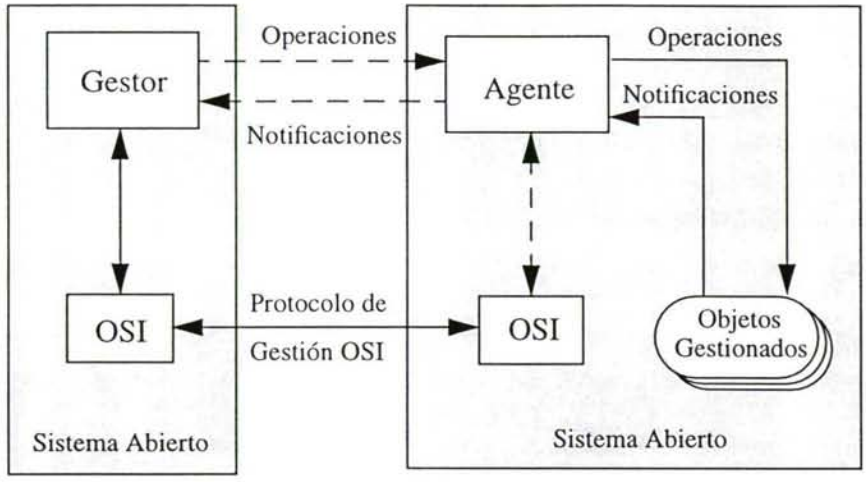

Figura 2.8. *Arquitectura del modelo OSI*

Un MO incluye atributos para ser usados como *relative distinguishing name* (RDN) que lo identifican unívocamente de otras instancias de su objeto padre en la MIT. Si se concatenan los RDN de los objetos que hay en el camino entre un objeto dado y el *root* del MIT se obtiene el *distinguishing name* (DN) que lo identifica unívocamente de cualquier otro objeto del MIT. Cada objeto gestionado es registrado en el árbol de registros de ISO mediante su OI (*Object IDentifier*), también es insertado en el nodo u hoja correspondiente dentro del árbol general de objetos que modelan un recurso (MIT). Por último, pueden establecerse ligazón de nombres (*name binding*) que permite establecer relaciones de continencia. De este modo el modelado de objetos en la familia OSI debe contemplarse desde estos tres puntos de vista, un árbol de nombrado, un árbol de registro y un árbol de continencia. Las interrelaciones entre objetos de gestión pueden expresarse a través de relaciones padre-hijo, o mediante ligazón de nombres. Los agentes (entidades gestionadas) y los gestores (entidades gestionadoras) son vistos como pares de aplicaciones que usan los servicios de los *Elementos de Servicio de Información de Gestión* (CMISE). Estos elementos utilizan los puntos de acceso al servicio (SAPs) para controlar las asociaciones entre gestor y agente. Las asociaciones se utilizan para el intercambio de información de gestión correspondiente a peticiones y respuestas, notificaciones e invocación de operaciones en los MO.

CMISE utiliza los protocolos OSI ACSE y ROSE [Rose, 90] para proporcionar estas funcionalidades. La estructura típica de esta arquitectura es la mostrada en la figura 2.9. Las instancias de los MO con sus atributos, operaciones y notificaciones están integradas en la MIT. El agente OSI proporciona funciones de selección para localizar los registros y acceder a través de servicios GET/SET/ACTION que le proporciona CMISE. También soporta la gestión de detección de eventos y su traslado a las entidades de gestión involucradas.

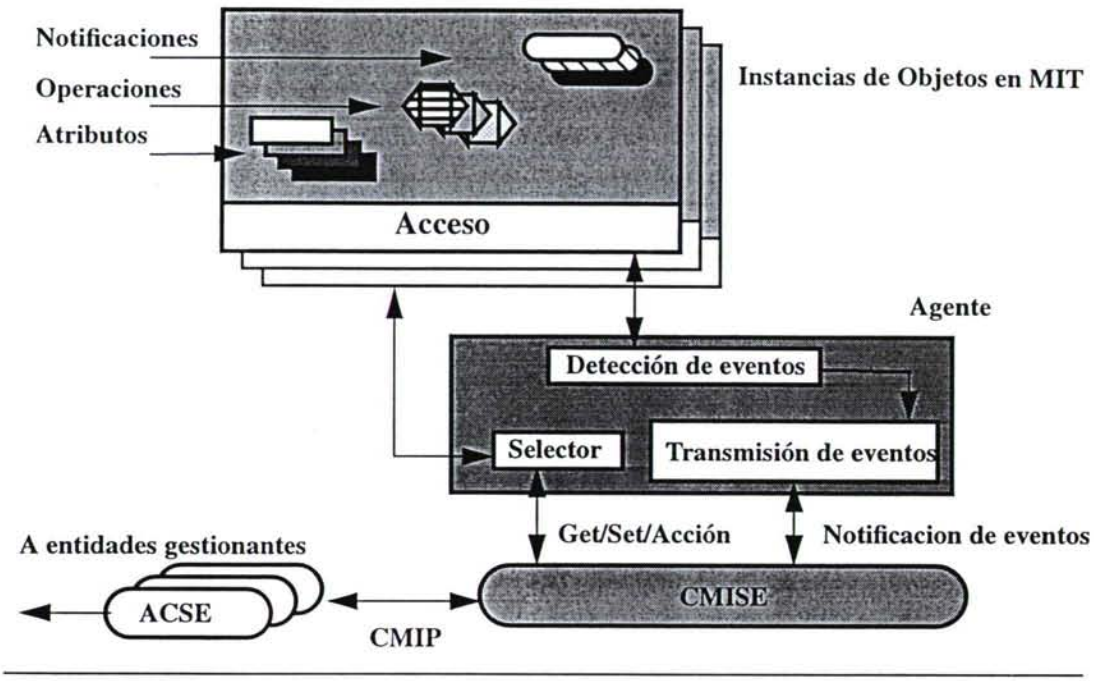

Figura 2.9. *Arquitectura el Modelo de Comunicaciones de Información de Gestión*

Los SAP que proporciona una entidad CMISE, mediante un transporte orientado a conexión, pueden agruparse de la forma descrita en la siguiente tabla:

Tipo	PDU	Función
Servicios de comunicación de gestión	M-INICIALIZE	Establece una asociación de gestión.
	M-TERMINATE	Finaliza una asociación de gestión.
	M-ABORT	Terminación no confirmada
Operaciones sobre el MIT	M-CREATE	Crea una instancia de un MO en el MIT
	M-DELETE	Borra una instancia de un MO en el MIT.
Servicios de manipulación de la MIT	M-GET	Recuperar / Obtener información.
	M-CANCEL-GET	Cancela peticiones de información.
	M-SET	Cambia el valor de un atributo de un MO
	M-ACTION	Invoca una operación sobre un MO
	M-EVENT-REPORT	Informa al gestor de un evento.

Tabla 5. *SAPs proporcionados por una entidad CMISE*

El núcleo de servicios de CMISE proporciona acceso a la información de gestión. Para identificar los objetos sobre los que se van a realizar las operaciones, se puede llevar a cabo *scoping*, que permite seleccionar el subárbol, niveles u objetos específicos. Una vez identificados estos objetos pueden realizarse operaciones de filtrado (*filtering*), operaciones *booleanas* que consisten en aseveraciones acerca de la presencia de ciertos valores de atributos sobre los objetos identificados.

Una vez realizadas estas dos funciones, es posible que se haya seleccionado más de un objeto sobre el que realizar la operación, el modelo de gestión OSI incluye el concepto de sincronización para indicar el orden en que los objetos serán procesados, de este modo, la operación se llevará a cabo si puede realizarse sobre todos los objetos (sincronización atómica) o sobre la mayoría (sincronización *best-effort*).

Como se ha indicado en la tabla anterior es posible llevar a cabo invocaciones a operaciones sobre instancias de MO indicando los parámetros oportunos, lo que proporciona una gran flexibilidad a este tipo de llamadas. Para ello se utilizan los servicios proporcionados por ROSE. También es posible crear o borrar instancias de MO en el MIT, que pueden utilizarse para ampliar la funcionalidad de la MIT. Un ejemplo es el uso de M-CREATE para instanciar en el MIT un objeto para gestión de notificaciones, de modo que cuando el agente reciba un evento utiliza el MIT para registrarlo, generando un M-EVENT-REPORT.

Dado que algunas de las peticiones de datos pueden desembocar en gran cantidad de datos que exceda el tamaño de las unidades de transporte, es necesario utilizar una estrategia de comunicación orientada a conexión, sin embargo, en ciertos entornos de trabajo esto puede resultar ineficiente ya que, además de aumentar la complejidad de gestores y agentes, puede perderse mucho tiempo y recursos en conexiones y restablecimiento cuando existe un pico de carga en la red. OSI cuenta con la primitiva M-CANCEL-GET para solucionar estas situaciones.

El objetivo del modelo OSI es maximizar el poder de modelado de la información para manejar sistemas complejos. Las diferencias fundamentales con respecto a la familia de protocolos *Internet* que veremos a continuación podemos resumirlas del siguiente modo:

- Las entidades gestionadas se modelan en base a objetos que encapsulan datos, relaciones, acciones y notificaciones de eventos. CMIP [ISO 9596] proporciona un rico y variado conjunto de procedimientos y paso de parámetros. La abstracción y herencia del modelo Orientado a Objetos ofrece importantes ventajas en el desarrollo de aplicaciones de gestión.
- La MIT constituye una base de datos dinámica, permitiendo una gran flexibilidad y eficiencia en el acceso a la información de gestión. Las entidades gestionadas pueden controlar el contenido y estructura de la base de datos y localizar sólo la información de interés. Sin embargo, esta estructura dinámica presenta grandes problemas de implementación, ya que los recursos necesarios para almacenar y procesar esta información no pueden predecirse en el momento del diseño. Es necesario plantearse en el diseño una disyunción entre funcionalidad y prestaciones de tiempo real sobre recursos limitados. Es en este balance, en donde, el modelo OSI ofrece dudas a los gestores de red.
- El modelo OSI permite la representación explícita de relaciones entre datos de gestión con el fin de potenciar la representación de un modelo Entidad-Relación.
- OSI introduce dos funciones no presentes en SNMP: el acceso masivo y selectivo. El elevado tráfico que genera CMIP puede reducirse realizando un acceso eficiente a los datos, mediante filtros o el acceso a múltiples datos similares a través de una única primitiva.

- OSI utiliza un transporte orientado a conexión lo que en situaciones críticas, el mantenimiento de las conexiones se hace especialmente difícil. Las entidades de gestión pierden tiempo y recursos en reinicializar conexiones perdidas. Eso puede convertirse en un obstáculo para llevar a cabo actividades de gestión.
- Las PDU's de CMIP suelen ser de gran tamaño debido al empleo de codificación BER (*Basic Encoded Rules*). Esta característica hace que el tráfico en la red sea muy superior al de otros estándares.

Dadas las características del modelo de gestión OSI, su utilización se hace necesaria en entornos complejos con equipos avanzados de telecomunicación, o bien, cuando las necesidades de operación son elevadas. En el caso de redes de datos, su utilización se pone en duda debido a la gran cantidad de recursos que consume su implementación.

2.2.4 Modelo centralizado basado en eventos: MINERVA

En [Hughes, 92] se presenta un modelo de gestión centralizado basado en eventos, denominado MINERVA. El modelo se centra en el principio de que una red puede entenderse como un conjunto de dispositivos entrelazados mediante un cableado o como un medio para proporcionar servicios a los usuarios finales. En este último caso, no es suficiente para la gestión de red, conocer el estado operacional del dispositivo, sino que es necesario conocer si el equipo está prestando un servicio a un nivel adecuado. Por otro lado, es muy complicado que el diseñador pueda prever de antemano toda la funcionalidad que va a desear del sistema. En este modelo, el usuario puede definir sus propios eventos con información y acciones asociadas. Esto permite ampliar dinámicamente la funcionalidad del sistema y facilita el desarrollo de un sistema flexible y extensible en términos de adquisición, interpretación, almacenamiento y visualización de la información. Otro de los aspectos fundamentales es que no se restringe a un protocolo de acceso a la información, como pe. SNMP, sino que ofrece un mecanismo simple para la gestión de dispositivos que no soportan protocolos estándar de gestión.

MINERVA facilita la personalización del sistema de gestión para adaptarlo a las necesidades reales del gestor. El principal esfuerzo de este proyecto se ha centrado en proporcionar un mecanismo mediante el cual el gestor de red pueda coleccionar información definida por el usuario de manera sencilla y eficiente. Esto abarca la representación y almacenamiento de la información definida por el usuario, incorporación de la información en el estado de la red que percibe el sistema de gestión de red y el método por el cual esta información puede ser introducida en el sistema de gestión.

El resultado ha sido un modelo central de interpretación y almacenamiento de datos conocido como *reporter*. El diseño del *reporter* está basado en que cualquier información que puede interesar a un sistema de gestión es el resultado de un evento. Es posible extender las capacidades de un sistema de gestión para que proporcione una serie de eventos definidos por el usuario. En la definición de cada evento debe introducirse información que facilite la correcta interpretación y procesamiento posterior del mismo. Cada notificación de evento, denominado *report*, tiene la capacidad de generar una entrada de *log* en el sistema de gestión, actualizar una relación en las bases de datos y generar una alerta en ese gestor. Esta actualización de la base de datos permite seguir un control en tiempo real del estado de la red. El *reporter* es únicamente un mecanismo mediante el cual la información capturada es registrada en la base de datos lo que facilita una

interface sencilla y consistente para la adquisición de información entre varios mecanismos de adquisición de datos, es decir, entre diferentes protocolos.

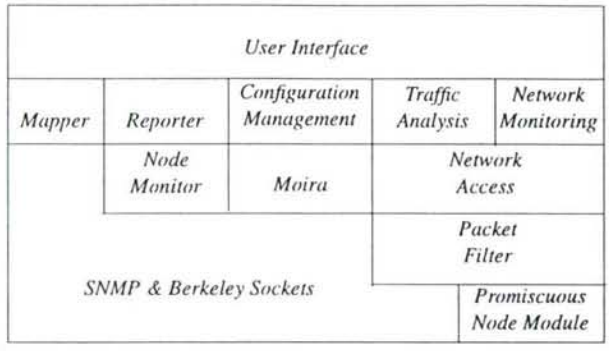

Figura 2.10. *Modelo en capas de Minerva.*

La figura 2.10 muestra la estructura del sistema Minerva, presentada en [Hughes, 93]. El modelo consta de cinco módulos de alto nivel, varios de bajo nivel y una interface de usuario que integra los módulos y proporciona un método consistente de interacción. Los cinco módulos de alto nivel citados son los siguientes:

- *Herramienta de mapeo de red:* Proporciona información topológica acerca de los elementos que componen la red a gestionar. Presenta una arquitectura de visualización jerárquica parecida a muchas plataformas de gestión existentes en el mercado.
- El *reporter*, citado anteriormente, permite emitir eventos con diferentes niveles de severidad de forma que puedan asociarse acciones a esos niveles. Tiene un submódulo denominado monitor de nodo que actúa de cliente del *reporter*. Monitoriza una lista de dispositivos chequeando el estado de los servicios proporcionados por el dispositivo.
- *Gestión de configuración*: Proporciona funciones de configuración de dispositivos y servicios que estos dispositivos pueden ofrecer.
- *Monitor de red*: Proporciona acceso a los datos intercambiados a través de la red. Permite al usuario ver e interpretar paquetes de datos intercambiados entre dispositivos. Incluye un lenguaje de reconocimiento de protocolos (PRL) que permite al sistema expandirse para reconocer nuevos protocolos.
- *Módulo de análisis de tráfico*: Utiliza la definición de protocolos en PRL para generar estadísticas acerca de los datos intercambiados en la red.

No es posible la integración de aplicaciones ya desarrolladas pues no tenemos ningún *gateway* o *middleware* que permita enlazar o llamar a rutinas de otras aplicaciones. El hecho de utilizar un lenguaje propietario (PRL) para el reconocimiento de paquetes de diferentes protocolos, lo hace en cierto aspecto cerrado, dado que el operador debe conocer este lenguaje para poder ampliar la funcionalidad del sistema. Expandir la funcionalidad del sistema resulta sencilla y flexible, siempre que el operador conozca los entresijos del lenguaje, aunque, por otro lado, también está limitado a este.

Al tratarse de un entorno centralizado adolece de ciertas desventajas inherentes: cuando el dispositivo en el que se encuentra el *reporter* falla, el sistema no puede funcionar correctamente. Si el número de monitores de red se incrementa considerablemente, es muy posible que se excedan las capacidades del *reporter* para analizar y procesar la información acerca de los eventos que está recibiendo. Por otro lado, la necesidad de enviar toda la información al *reporter* sin que los monitores de red, procesen parte de esa información hace que se incremente sustancialmente el tráfico en la red, con la consiguiente degradación del rendimiento general de la red. En cuanto al acceso a la información de gestión, se realiza mediante un acceso al gestor de base de datos *Ingres*.

Este sistema puede utilizar *reporters* intermedios que preprocesan los eventos enviando a un nodo central únicamente aquellos que le interesan.

2.2.5 Modelos basados en DBMS: MANDATE

La base de datos de información de gestión es el núcleo de los actuales sistemas de gestión y resulta un punto critico de su implementación, debido a que todas las funciones de gestión interactuan sobre ella. Algunas plataformas aumentan la funcionalidad de los sistemas de gestión mediante la utilización de modelos estándares de gestores de bases de datos. En este apartado nos centraremos en una representativa: MANDATE (*MAnaging Networks using DAtabase TEcnology*) [Haritsa, 93]. El principio que guía el desarrollo de MANDATE es que el operador de red interactúa únicamente contra una base de datos, cualquier cambio que necesite realizar sobre la red, se traduce en actualizaciones sobre variables de la base de datos. Esta base de datos (MIB) ha sido desarrollada siguiendo unos principios arquitecturales (tolerancia a fallos, escalabilidad, *triggers*), requerimientos de las interfaces (homegeneización, navegación), requerimientos de automatización (mecanismos activos, toma de decisiones, control), y finalidades de rendimiento. La información de gestión clasificada en tres categorías (ver figura 2.11), datos sensor, datos estructurales y datos de control. Los datos sensor representan el estado de la red, los datos estructurales describen la construcción física y lógica de la red, mientras que los datos de control almacenan los campos operacionales de la red.

Los cambios en la red física deben llevarse a cabo mediante la ejecución de procesos que son activados o disparados por el sistema de base de datos. Por tanto, MANDATE es un sistema orientado a eventos, pero que actúa de manera estática frente a estos. Como lenguaje interrogador de la base de datos se utiliza una interfaz orientada a objetos compatible con el modelo OSI, pero que utiliza en sus capas de almacenamiento físico un modelo relacional. Además es posible definir vistas de red (partes del árbol de datos de la MIB). Es posible el acceso a la MIB desde diversas partes de la red, dado que se implementan los servicios CMIS para transmitir y recibir peticiones a través de CMIP. También es posible el acceso a través de otros protocolos como SNMP, sin más que realizar el correspondiente puente.

Otra de las características de MANDATE es que si bien, se trata de una base de datos centralizada con toda la información de gestión, su implementación física en redes de gran tamaño, puede ser distribuida por razones de rendimiento. Se sigue un modelo cliente/servidor tradicional para llevar a cabo esta implementación. Sin embargo, esta aproximación puede crear problemas de rendimiento en los servidores, en lo que se refiere a capacidad de procesamiento y E/S. Esto se soluciona añadiendo funcionalidad en los clientes para que puedan gestionar y procesar pequeñas vistas de la MIB.

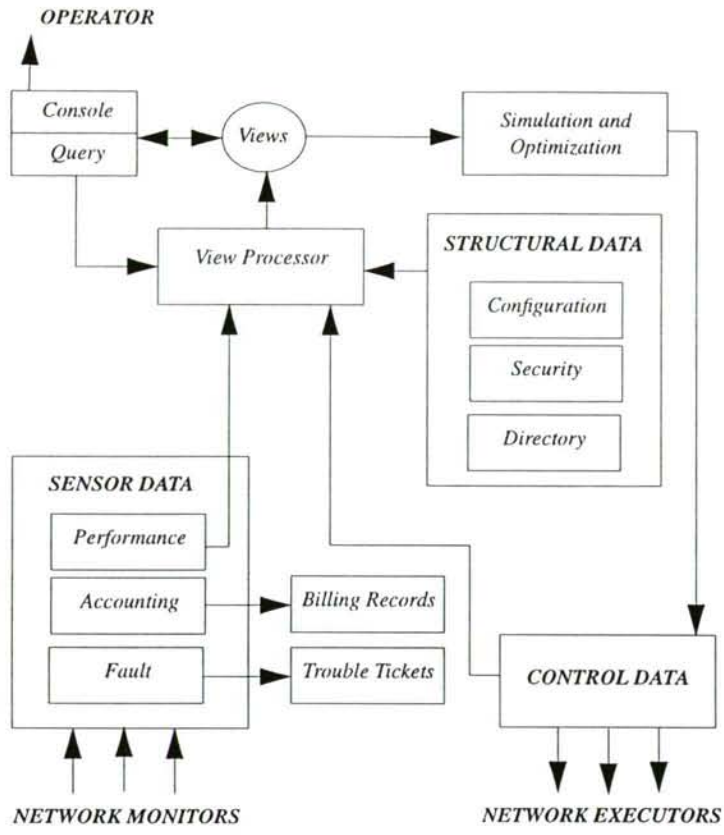

Figura 2.11. *Modelo de MIB en el sistema MANDATE*

El uso de una base de datos propietaria hace que el sistema no resulte sencillo de manejar, pudiendo resultar difícil de configurar. Además no es sencillo expandir la funcionalidad del sistema de gestión, para ello es necesario modificar la MIB y sus acciones asociadas por lo que el coste de expansión resulta elevado. Si el número de clientes se incrementa considerablemente el número de peticiones sobre la base de datos centralizada puede conllevar problemas de rendimiento sobre el servidor. Los clientes disponen de una relativa autonomía dado que fallos en el servidor de base de datos hace que todo el sistema deje de funcionar. El procesamiento se encuentra centralizado y el conjunto de acciones que los clientes pueden llevar a cabo es muy limitado. Aunque MANDATE ha sido desarrollada para interactuar con protocolos de gestión estándar, la implementación de la MIB que realiza no es del todo estándar.

El uso de una base de datos facilita la seguridad de información, tanto en acceso como en persistencia, siendo posible desarrollar MIB con información detallada acerca de los eventos que ocurren en la red. Es posible definir varias pasarelas desde protocolos estándar o propietarios a interfaces de base de datos para trabajar con el sistema. También es posible que aplicaciones ya desarrolladas integren esta base de datos para trabajar con ella.

2.3 Modelos jerárquicos

Una variación de la arquitectura centralizada es la basada en una plataforma jerárquica [Terplan, 92], que se muestra en la figura 2.12. En este caso, el gestor se divide en dos partes: la plataforma de gestión y las aplicaciones de gestión. La plataforma realiza un primer paso en el procesamiento de los datos de gestión: recolección, habilitación de servicios de gestión como monitorización y control, cálculos de rendimiento, etc., ocultando los protocolos de gestión subyacentes para ofrecer una vista abstracta de la gestión a las aplicaciones, las cuales operan en un segundo nivel de procesamiento en funciones de toma de decisiones y funciones de alto nivel para cálculos específicos. Estas dos partes se comunican mediante *Common Aplication Programming Interface* (API) que son un conjunto de librerías cuyas funciones pueden ser invocadas por las aplicaciones para acceder al *Kernel* de la plataforma de gestión.

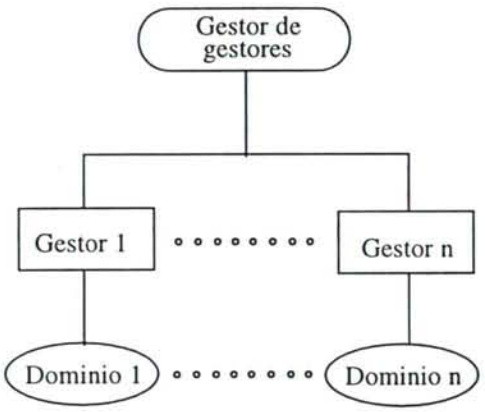

Figura 2.12. *Arquitectura jerárquica..*

Esta arquitectura ofrece un fácil mantenimiento, mejora la capacidad de expansión y simplifica el desarrollo de aplicaciones en entornos heterogéneos. La heterogeneidad y la complejidad de los protocolos de bajo nivel es responsabilidad de la plataforma, ocultando estas características a las aplicaciones. Sin embargo, presenta el problema de cualquier entorno centralizado, la limitada escalabilidad, la plataforma puede convertirse en un serio cuello de botella cuando el número de aplicaciones se incrementa sustancialmente.

En una arquitectura jerárquica se utilizan múltiples sistemas de gestión, cada uno con un sistema actuando como servidor y varios actuando como clientes. Algunas de las funciones de la plataforma de gestión residen en el servidor mientras que otras lo hacen en los clientes. La plataforma utiliza una base de datos con arquitectura cliente/servidor. Los clientes pueden no tener sistemas de bases de datos separados pero utilizan comúnmente el sistema de base de datos del servidor a través de la red. Suelen realizarse respaldos del servidor para garantizar la estabilidad del sistema.

Esta aproximación alivia uno de los principales problemas de la gestión centralizada al distribuir las tareas de gestión entre el servidor y los clientes. Los clientes, en condiciones severas pueden no necesitar toda la funcionalidad del gestor central. Dado que no existe una localización centralizada de toda la información de gestión, puede que la obtención de esta información por parte del operador sea costosa en tiempo. Además la lista de dispositivos que puede gestionar un cliente debe ser lógicamente predeterminada y con una configuración manual.

La arquitectura jerárquica también aplica el paradigma de los dominios de gestión e introduce el concepto de "*gestor de gestores*". Cada gestor de un dominio es responsable únicamente de la gestión de ese dominio y no del resto de la red. El gestor de gestores (MOM) opera en un nivel jerárquico más alto, solicitándo información a los gestores de dominios. En esta caso, no existe comunicación entre los gestores de dominios directamente. Esta arquitectura es facilmente escalable hacia arriba, añadiendo más MOM (MOM de MOMs), esto nos lleva a un esquema de múltiples niveles jerárquicos. Otra ventaja de esta organización es que ofrece facilidades para el desarrollo de aplicaciones que recuperen información de múltiples dominios.

2.3.1 Modelo OSIMIS

OSIMIS [Pavlou, 95] (*OSI Management Information Service*) es una plataforma que proporciona la funcionalidad necesaria para la construcción de sistemas complejos de gestión de manera fácil, rápida y eficiente. Está basada en el modelo de programación orientado a objetos, basada en OSI [ISO 10040] e implementada en C++ [Stroustrup, 86]. Proporciona un entorno para el desarrollo de aplicaciones de gestión ocultando los detalles de los servicios de capas inferiores mediante APIs de alto nivel. De este modo, consigue que los diseñadores se centren más en la inteligencia de la aplicación de gestión, que en el acceso a los servicios de bajo nivel y protocolos. OSIMIS, también ha sido desarrollado para facilitar la integración de sistemas existentes donde cohabiten facilidades de gestión propietaria con diferentes modelos de gestión estándar. Se han realizado puentes que permiten trasladar peticiones SNMP en peticiones CMIP [Pavlou, 93]. La plataforma no sigue el modelo tradicional agente-gestor, una aplicación puede desempeñar ambos papeles dependiendo del nivel jerárquico en que se encuentre.

OSIMIS, ver figura 2.13, utiliza ISODE (*ISO Development Environment*) [Rose, 91] como mecanismo de comunicación OSI de bajo nivel pero también pueden llevarse a cabo implementaciones basadas en XOM/XMP [XOpen, 92]. ISODE presenta grandes ventajas, ya que implementa servicios como FTAM (*File Transfer Access and Management*) y la incorporación del servicio de directorios X.500 de OSI, que son fundamentales en entornos de gestión complejos.

OSIMIS es una plataforma que cuenta con los siguientes servicios:

- APIs de alto nivel orientadas a objetos e implementadas como librerías, en las que se implementan los protocolos y herramientas estándares: CMIP, SNMP y ASN.1.
- Aplicaciones genéricas como *browsers*, pararelas, servidores de directorios, etc.
- Un mecanismo de coordinación que permite estructurar una aplicación de forma totalmente dirigida a eventos y extenderla para intercomunicarse de la misma manera.

Otro de los aspectos interesantes de OSIMIS es la gestión de eventos. Las aplicaciones distribuidas necesitan integrar eventos que le llegan tanto de dentro como del exterior, dando cumplida respuesta a los mismos. En general existen dos formas de afrontar el manejo de estos eventos internos y externos:

- a.- Utilizar un paradigma de ejecución *mono-thread.*
- b.- Utilizar un paradigma de ejecución *multi-thread.*

En el primer modelo, las comunicaciones externas deben seguir un modelo asíncrono, esperando por el resultado de una operación remota de manera sincrona que bloquea totalmente el sistema. Se necesita un mecanismo que permita preparar y demultiplexar todos los datos pedidos exteriormente, esto es parte del soporte de coordinación de OSIMIS. En el segundo caso, muchos *threads* pueden ser ejecutados simultáneamente por el mismo proceso. Este es el estilo de programación utilizado en plataformas de sistemas distribuidos que se basan en el protocolo RPC [RPC, 84] que es inherentemente síncrono con respecto a clientes que realizan operaciones remotas sobre servidores.

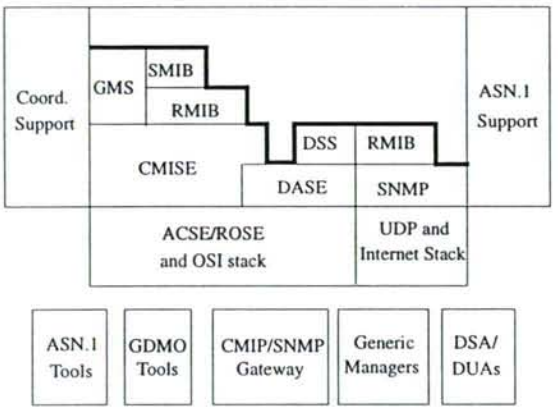

Figura 2.13. *Arquitectura OSIMIS.*

Un problema adicional, es el manejo de alarmas temporales internas, muchos sistemas operativos no permiten apilar varias alarmas, permitiendo una única alarma pendiente por proceso. OSIMIS proporciona una infraestructura orientada a objetos que permite organizar una aplicación de manera totalmente orientada a eventos siguiendo el paradigma de ejecución *mono-thread.* Cualquier evento externo o interno es serializado y atendido de manera FIFO (*first-come-first-served*). Este mecanismo permite la fácil integración de fuentes adicionales externas de entradas o alarmas temporales y se realiza mediante clases C++: el *Coordinator* y el *Knowledge Source* (KS). Como indica la figura 2.14 en cada aplicación habrá un *Coordinator* y varios DS que permiten el uso de los servicios del *Coordinator* integrando fuentes externas de entradas o alarmas temporales. Todos los eventos y alarmas temporales externas son controladas por el *Coordinator* cuya presencia es transparente a las implementaciones de los KS específicos a través de un interfaz abstracto para los KS.

El *Generic Managed System* (GMS) [Pavlou, 93b] integrado en OSIMIS, proporciona soporte para la construcción de agentes que ofrecen totalmente la funcionalidad OSI, incluyendo *scoping*, *filtering*, control de acceso, respuestas enlazadas y cancelación de *gets*. Además, OSIMIS soporta totalmente las funciones de gestión de objetos, emisión de eventos, control de *logs*, control de alarmas y control de acceso.

A modo de resumen, se puede citar que OSIMIS es de difícil manejo, dado que si bien cuenta con numerosas herramientas de diseño de aplicaciones, su utilización no es simple y su integración en una única plataforma de gestión no resulta sencilla. Por otro lado, aunque relacionado con lo anterior, la configuración de estas aplicaciones no es sencilla para los operadores de red. La informa-

ción de gestión no se encuentra distribuida por la red, sino que reside toda en la plataforma de gestión, sin embargo, la utilización de un modelo jerárquico posibilita la distribución de la información global entre varios gestores. Todo el procesamiento de gestión se realiza en el gestor, aunque puede delegar parte de estas tareas en gestores intermedios, sin embargo no se consigue una distribución real del procesamiento y aumenta el tráfico de red, con el necesario intercambio de información entre gestores intermedios. Por otro lado, los agentes presentan una limitada autonomía, resultando inoperables cuando la red está caída. En cuanto a mecanismos de seguridad, OSIMIS permite autenticación de entidades, autenticación e integridad de datos y control de accesos.

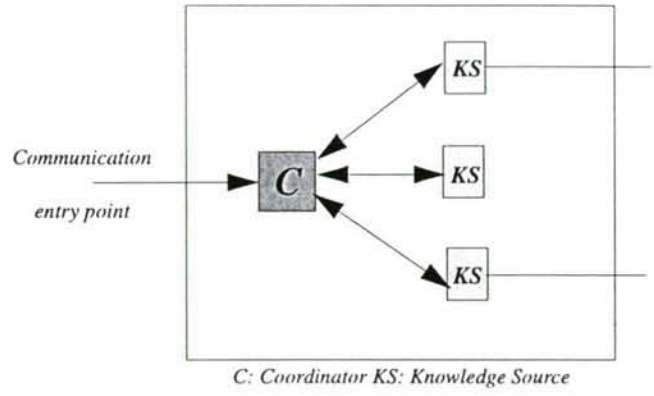

Figura 2.14. *Modelo de agente OSIMIS.*

Sin embargo, presenta grandes ventajas en cuanto a la flexibilidad de expansión (aunque con las limitaciones de un entorno centralizado), un relativo bajo coste de expansión (siempre y cuando el operador pueda desarrollar aplicaciones con las herramientas que incorpora OSIMIS). También presenta una buena escalabilidad, dado que el número de agentes y gestores se puede incrementar considerablemente debido a su estructura jerárquica. En cuanto a la información la utilización del modelo OSI consigue un entorno de información seguro, con cierta riqueza semántica y con la posibilidad de accesos masivos a la información de gestión de que se dispone. Resulta sencillo realizar puentes entre protocolos (algunos ya se han implementado, como el *SNMP-to-CMIP*) y el uso de facilidades de integración de aplicaciones como CORBA puede facilitar la integración de aplicaciones dentro del sistema de gestión. Por otro lado, OSIMIS es una plataforma orientada a la estandarización de los protocolos y la información que utiliza.

2.3.2 Modelo jerárquico basado en plataformas: NETMODE

Un modelo basado en plataforma no es la mejor arquitectura para escalar eficientemente un sistema de gestión, cuando la red crece en tamaño y tráfico. El tiempo de respuesta de procesamiento de una plataforma se incrementa proporcionalmente según la red va aumentando de tamaño. El incremento del numero de aplicaciones que son implementadas sobre la plataforma puede causar un efecto de cuello de botella, decrementando el rendimiento del sistema. Con vistas a mejorar la escalabilidad, sin sacrificar la flexibilidad de la aproximación basada en una plataforma, [Stama, 95] propone una arquitectura con múltiples gestores y con forma de plataforma de gestión. Este modelo, conocido como servidor plataforma, está basado en las comunicaciones gestor-gestor. Cada gestor ofrece un conjunto de servicios de gestión que pueden ser usados tanto por aplicaciones como por otros gestores. En esta arquitectura se distingue entre:

- ***Entidad gestora***: Puede ser un gestor de un dominio, un gestor de gestores o un gestor integrado. El gestor de un dominio proporciona información y es responsable de un determinado dominio de gestión. Un gestor de gestores maneja a un grupo de gestores de dominios y proporciona información sumarizada de los mismos. Por otro lado, un gestor integrado es responsable de un dominio que incluye otros gestores y proporciona información acerca del dominio y de los sumarios emitidos por los gestores.
- ***Entidad aplicación***: Puede ser una aplicación local o una aplicación integrada. Una aplicación local es aquella que es implementada sobre cualquier gestor simple, mientras que una integrada es la que usa los servicios de más de un gestor.

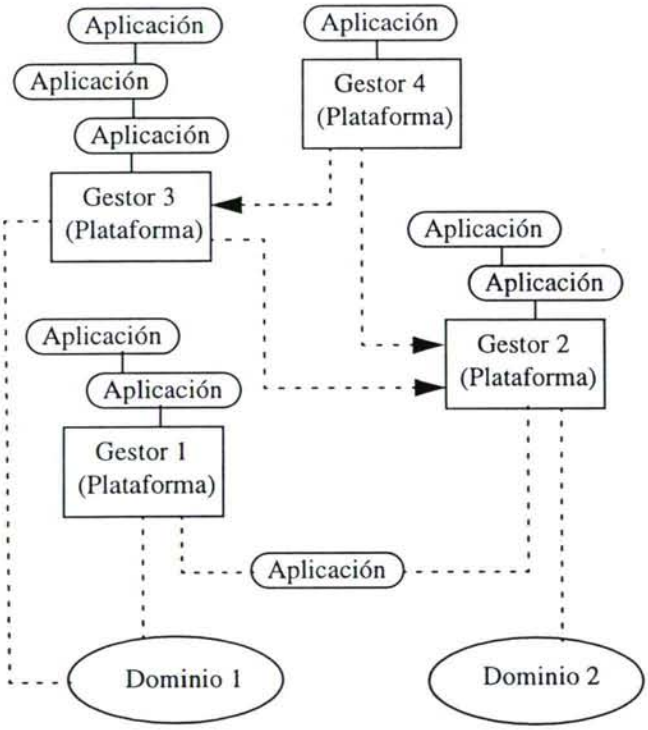

Figura 2.15. *Escenario típico de una arquitectura platform-server*

La figura 2.15 muestra un escenario típico de este tipo de arquitectura. Los gestores 1 y 2 son gestores de dominios, responsables de los dominios de red 1 y 2 respectivamente. El gestor 4 es un gestor de gestores, responsable de los gestores 2 y 3 haciendo uso de los servicios de estos. Finalmente, el gestor 3 es un gestor integrado que gestiona el dominio 1 y el gestor 2. Varias aplicaciones locales operan en la cima de cada gestor, mientras que una única aplicación integrada utiliza los servicios de los gestores 1 y 2.Cada una de las plataformas de gestión propuestas, está descompuesta en tres niveles (ver figura 2.16):

- *Interfaz de red y protocolo de gestión*: Gestionan las comunicaciones con otros elementos de gestión de capas inferiores en la jerarquía. Estos elementos pueden ser agentes propietarios, SNMP, SNMPv2, CMIP u otras plataformas de gestión. Este

módulo ofrece también una interfaz para otros protocolos o servicios de red, como el caso el ICMP que puede ser utilizado para monitorizar el estado de diversos componentes.

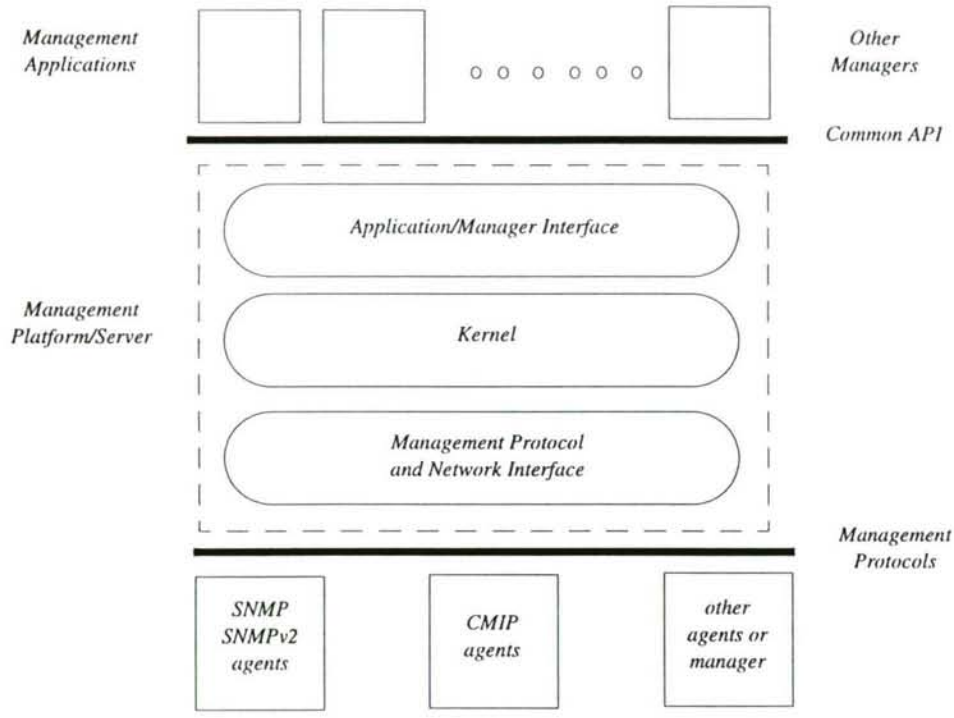

Figura 2.16. *Arquitectura por capas de NETMODE.*

- *Kernel*: Este módulo implementa las funciones primarias de gestión. Utiliza los servicios de las capas inferiores para recuperar información de la red y proporcionar servicios a las aplicaciones y a otros gestores a través de su capa más alta. Este módulo se descompone en submódulos como el gestor de *log*, gestor de alarmas, gestor de seguridad y contabilidad, gestor de objetos, servicios de monitorización de MIB, gestor de eventos, servicios de monitorización de estado, servicios gestor-gestor. La relación entre estos submódulos y las funciones de gestión (SMF) definidas en OSI puede observarse en la siguiente tabla:

Submódulo del Kernel	SMF OSI asociada
Gestor de *log*	Log Control
Gestor de alarmas	Alarm Reporting, Workload Monitoring
Gestor de seguridad y contabilidad	Security Audit Trail, Security Alarm Reporting, Access Control, Accounting Meter
Gestor de objetos	Object Management, Relationship Management, State Management, Event-Report Management
Servicios de monitorización MIB	Workload Monitoring
Gestor de eventos	Event-Report Management

Tabla 6. *SMFs OSI asociadas a los submódulos del Kernel*

Submódulo del Kernel	SMF OSI asociada
Servicios de monitorización de estado	State Management
Servicios gestor-gestor	Summarization, Alarm Reporting, Event-Report Management

Tabla 6. *SMFs OSI asociadas a los submódulos del Kernel*

- *Interfaz Aplicación/gestor:* Gestiona todas las comunicaciones con las aplicaciones y gestores que utilizan los servicios de una plataforma. Este módulo implementa un agente que utiliza una *Manager MIB*, propuesta en [Stama, 95b] para proporcionar los servicios de la plataforma.

El uso de una *Manager MIB* permite organizar las arquitecturas basadas en plataformas con configuraciones jerárquicas, de modo estandarizado, unificando el acceso por parte de plataformas y aplicaciones.

Por lo tanto, NETMODE es una plataforma con múltiples gestores, para una gestión jerárquica de múltiples dominios de gestión y basada en un núcleo de información central compuesta por una base de datos. Se define la funcionalidad de gestión mediante servicios de gestión que son aplicados sobre objetos de una base de datos y se implementa siguiendo un RDBMS convencional, al que se puede acceder por medio de una interfaz CORBA. Esto facilita la integración de protocolos y aplicaciones en el sistema.

El uso de múltiples gestores y múltiples aplicaciones de distinto tipo hace que el sistema resulte complejo de configurar y manejar. Además el mapeo de servicios sobre objetos de la base de datos hace que el sistema sea estático, y que el incremento de funcionalidad resulte complicada y costosa.

La utilización de una estructura jerárquica de gestores hace que el sistema resulte fácilmente escalable aunque a costa de sacrificar el rendimiento global de la red, debido al necesario intercambio de información entre gestores.

El uso de una base de datos para almacenar la información y los servicios de gestión hace que el sistema resulte seguro y accesible. Sin embargo, la centralización de la información hace que si bien, se distribuye el procesamiento entre los gestores, la información de gestión permanece centralizada pudiendo condicionar la operación de gestores de otros niveles.

2.3.3 Modelo NAS

El centro de investigación Ames de la NASA ha propuesto un sistema jerárquico propietario denominado HNMS (*Hierarchical Network Management System*) que proporciona una arquitectura escalable para la monitorización de grandes redes.

La arquitectura del modelo HNMS está compuesta por cuatro módulos:

- *Server Module* (SM): Núcleo del sistema de gestión, proporciona al resto de módulos información acerca de la topología de la red e información acerca de su estado.
- *User Interface Module* (UIM): reside en una estación gráfica proporciona al usuario un acceso en tiempo real a los datos.

- *Input/Output Module* (IOM): reside en hosts estratégicamente colocados en la red. Estas máquinas son las encargadas de recolectar datos de gestión. Utilizan para ello SNMP. Estos módulos filtran la información que deben pasar al SM, el cual, a su vez, únicamente envía esta información a los módulos UI participantes en el sistema de gestión. Con esta aproximación se consigue una mejor distribución del tráfico.
- *Database Module* (DBM): En este módulo reside la información de respaldo que permite al *server* analizar y estudiar estos datos. Si un *server* deja de funcionar, cuando se recupere o bien se reinicie desde otro lugar, puede cargar de esta base de datos el estado de la red.

Cada objeto representado por el sistema de gestión tiene un estado que representa e identifica el estado operacional y administrativo de los elementos de la red. También se representan relaciones entre objetos. La comunicación entre módulos es realizada mediante un protocolo propietario (*Hierarchical Network Management Protocol*) HNMP construido sobre UDP/IP. Este protocolo necesita una nueva MIB denominada HNMS MIB que define el conjunto de variables que define los objetos representados. Este protocolo permite a su vez facilidades de alto nivel como la gestión de dominios y la subscripción. Esto va a permitir enviar o recibir datos de grupos de objetos de una única vez, identificar univocamente los datos de un objeto dado. El HNMS utiliza por tanto cuatro niveles: agente en el dispositivo, módulo IO, *server* y módulo IO o bien DBM.

Cada *server* y módulos IO se ejecutan como demonios en máquinas separadas. Cada *server* se distingue por un nombre de comunidad (concepto diferente al de comunidad SNMP) que establece los parámetros administrativos y de autenticación en cuanto al acceso al mismo.

El módulo IO inicia las comunicaciones con el *server* mediante un mensaje "*hello*", a lo que el *server* responde con un "*Wellcome*", asignándole un módulo y su identificador (instancia del *server*). Una vez comenzada la conexión, se negocia la posición actual del *server*. Se utiliza un esquema de ventana deslizante a la hora de controlar las conexiones.

La interfaz de usuario de HNMS ha sido desarrollada basándose en los siguientes requerimientos:

- Proporcionar automáticamente pantallas que indiquen los cambios que se producen en la red. Todos los elementos de la red son representados individualmente.
- Proporcionar una visión general y actual del estado de la red sin comprometer para ello el rendimiento de la misma.
- Proporcionar la notificación de los eventos más importantes al usuario sin exponerle información irrelevante.
- Proporcionar información detallada acerca de cada elemento de la red.

Para esto último se han implementado cuatro tipos de diagramas de estado, estos diagramas son actualizados por el *server* cuando varia:

- *Wan diagram*: Permite visualizar el aspecto de la red incluyendo los *routers* que forma la WAN.
- *Site diagram*: Puede ser creado al seleccionar un lugar concreto de la WAN. Representa todas las LANs que están conectadas a un *router* en concreto.
- *Custom diagram*: Permite al usuario construir vistas personalizadas con conjuntos de elementos de red que pueden ser observados.

- *Object diagram*: Indica para cada elemento como se modifican sus variables. Es un diagrama de bajo nivel que permite al usuario obtener información pormenorizada de los elementos que están siendo gestionados.

El sistema es sencillo de manejar, aunque la existencia de múltiples niveles de gestión complica la configuración del mismo. El uso de múltiples niveles también incrementa la facilidad de expansión y escalabilidad del sistema. Sin embargo, puede dañar seriamente el rendimiento de la red. Por otro lado, el incremento de funcionalidad del sistema implica la reconfiguración de muchas partes del mismo por lo que su coste es elevado.

En cuanto a la fiabilidad del sistema, esta es limitada. Si un *server* se cae o no es alcanzable, se arranca un nuevo *server* en otra máquina de la red. Además, si la base de datos no puede ser accedida, se encolan las peticiones y datos hasta que se pueda restaurar el servicio.

No es posible integrar con el sistema nuevos protocolos ni nuevas aplicaciones dado que utiliza el modelo estándar SNMP para consulta de la base de datos. Además el uso de MIBs SNMP hace que la riqueza semántica de la información de gestión sea relativamente pobre. El acceso a la misma, también está limitado por el uso de este lenguaje de interrogación que no permite actuar sobre estructuras complejas de datos. Además, la información y el procesamiento son centralizados por lo que el rendimiento global de la red cae sustancialmente cuando la red crece en tamaño y funcionalidad. Otro de los inconvenientes de la utilización de SNMP es que la seguridad en el acceso a la información de gestión es muy pobre, dado que el protocolo no dispone de métodos seguros para acceder a la información.

2.3.4 Agentes intermedios: Midlevel managers

El incremento del tamaño de las redes nos lleva a enfoques jerarquizados o distribuidos de gestión de red. El uso de *midlevel managers* parece solucionar este problema. En [Siegl, 94] se propone un proceso denominado SubAgente (*SubManager*) que actúa de intermediario entre el gestor y el agente. Este proceso es colocado al lado de cada agente que va a ser gestionado por un gestor. La microgestión llevada a cabo por el *submanager*, reduce el flujo de información de gestión entre el gestor y los agentes.

La comunicación gestor-gestor descrita en SNMPv2 es un método rígido para solucionar los problemas del paradigma centralizado. Un gestor de un nivel superior indica a uno subordinado para que de servicio a un determinado agente. Este gestor sólo puede monitorizar información contenida en la MIB de este agente. Con esta aproximación no es posible monitorizar expresiones complejas. Así, por ejemplo, no es posible chequear la relación entre todos los valores recibidos de una determinada variable que ha desencadenado un error. Disponer de un gestor chequeando gran multitud de valores almacenados en las MIBs de los agentes no es la mejor opción, deben considerarse otras alternativas, como el caso del *SubManager* que actúa entre el agente y el gestor. Esta herramienta asigna dinámicamente nuevos procesos de gestión de red (NMPs) cada vez que el operador requiere más o mejor información. La configuración dinámica es una característica esencial de este tipo de sistemas, dado que no todas las operaciones de gestión pueden ser previamente definidas en tiempo de compilación del agente. Periódicamente el *SubManager* realiza cálculos con los valores obtenidos del agente para posteriormente ofrecérselos al gestor principal cuando los requiera.

La figura 2.17 muestra la arquitectura general de un *SubManager* que sigue el siguiente proceso de operación. En primer lugar el gestor carga el procedimiento de gestión de red (NMP) y se lo asigna al correspondiente *SubManager*. El NMP es almacenado en el propio *SubManager* en dos tablas diferentes, la *subMgrEntry* y la *subMgrOps*. Este procedimiento es especificado en un lenguaje de programación propietario y para su traslado al *subManager* se han desarrollado herramientas que permiten realizar estas tareas a las plataformas de gestión estándar. Si se activa el proceso se genera o crea un *worker* que realiza los cálculos o peticiones periódicas sobre el agente. Este proceso evalúa el NMP y almacena los resultados en la tabla *subMgrValue* que es compartida por el *worker* y el *controler*. Si el gestor necesita información acerca de la evaluación llevada a cabo por el NMP, lee los valores almacenados en la tabla *subMgrValue*. Es posible cargar muchos NMPs en el *subManager*, que es capaz de controlar (con un único proceso *controler*), varios procesos *worker*. La MIB de resultados puede ser consultada por cualquier otro gestor mediante el protocolo de comunicaciones SNMP. Adicionalmente al almacenamiento de los valores resultado de la evaluación de un NMP, también se definen una serie de valores históricos por parte del usuario que permiten realizar cálculos diferenciales con respecto a ciertas variables de la MIB.

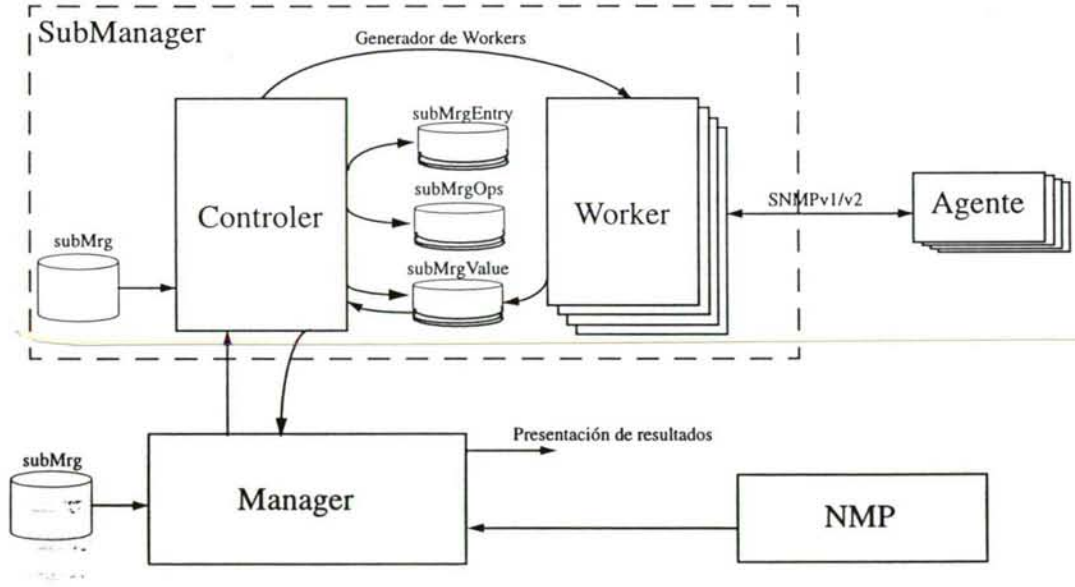

Figura 2.17. *Arquitectura de un SubManager*

El *submanager* es un elemento intermedio que se puede acoplar a cualquier plataforma estándar de gestión, por lo que en principio no plantea problemas de configuración y utilización. Además, la funcionalidad del sistema puede ser incrementada sustancialmente con la adicción de nuevos procedimientos de gestión (NMPs). Sin embargo, estos procedimientos de gestión tienen que ser desarrollados en un lenguaje propietario, por lo que se pierde en estandarización y apertura de la plataforma. El uso de SNMP para el acceso a la MIB del agente acarrea todos los problemas inherentes a este protocolo, limitación de acceso a la información, incapacidad para manejo de datos complejos, falta de mecanismo de seguridad (aunque en algunos casos se utiliza SNMPv2) y de control de acceso, etc. Por otro lado resulta imposible integrar nuevos protocolos y acceder a aplicaciones previamente desarrollados en otro lenguaje. Si embargo, el procesamiento puede considerarse distribuido, dado que muchas de las operaciones de gestión son realizadas en el propio subagente. Además la fiabilidad del agente aumenta al decrementarse la dependencia con respecto al gestor. La escalabilidad del sistema es relativa dado que el núcleo del sistema presenta

una arquitectura claramente centralizada con lo que se ve afectado el rendimiento del sistema global.

2.4 Modelos distribuidos

La aproximación distribuida se asocia con el concepto de gestión de dominios, utilizando más de un gestor para el control de la red según unas directrices prefijadas basadas en razones geográficas, de organización, etc. Cuando un gestor necesita información de otro dominio de gestión se comunica con el gestor responsable del mismo para obtenerla. La escalabilidad es la gran ventaja de este tipo de aproximaciones. Requerimientos de alto rendimiento del sistema o posibles expansiones de la red gestionada, se satisfacen mediante la creación de nuevos dominios de gestión, añadiendo los necesarios gestores. Esta arquitectura es presentada en la figura 2.18.

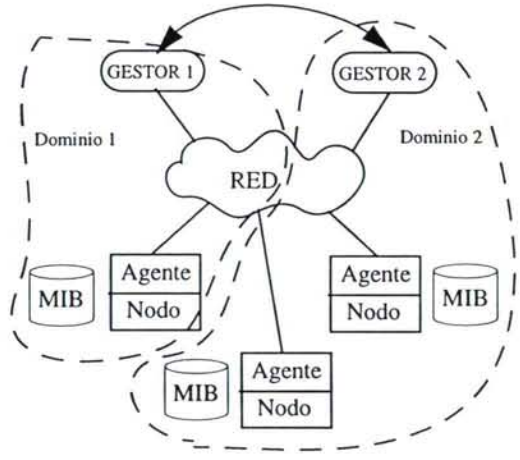

Figura 2.18. *Arquitectura distribuida.*

El entorno de gestión distribuido utiliza múltiples plataformas o sistemas con las mismas funcionalidades. Se agrupan en conjuntos liderados por una de ellas. Cada una de las plataformas individuales tiene una base de datos completa de los dispositivos que forma la red, lo que permite realizar varias tareas y enviar resultados a un sistema central.

Las ventajas de este tipo de arquitectura son las siguientes:

- Localización centralizada de toda la información de red, alarmas y eventos, así como el acceso a las aplicaciones o funciones de gestión
- Independencia de un gestor central. Si uno de los gestores falla o esta sobrecargado, cualquier otro gestor de la red o incluso el propio agente, puede tomar el control y realizar estas funciones.
- Distribución de las tareas de gestión, muchas de las funciones son realizadas sobre los agentes que están manejando los dispositivos. Es decir, la información es procesada allí donde se produce.

La tecnología de servidores de replicación de bases de datos es especialmente útil en este tipo de sistemas. Un servidor de replicaciones controla varias bases de datos en diferentes lugares de

manera sincronizada. Sin embargo, la sobrecarga introducida para obtener esta sincronización puede consumir una cantidad de recursos considerable.

Algunos de los problemas de los productos comerciales que pretenden realizar herramientas distribuidas para gestión de red, son los siguientes:

- Uso de protocolos de comunicación propietarios entre los gestores, por lo cual no soportan entornos heterogéneos multi-vendedor.
- Los agentes inteligentes pueden realizar procesamiento de información de alto nivel obtenida del propio agente, pero no de varios agentes.
- La mayoría de las veces, es necesario definir MIBs privadas para cada uno de los posibles resultados de los procedimientos de gestión distribuidos.

A continuación abordaremos algunas de las arquitecturas distribuidas más usuales.

2.4.1 Agentes extensibles con código en MIBs

En [Williamson, 96], se introduce un paradigma distribuido basado en agentes que pueden extender su funcionalidad, incorporando código almacenado previamente en una MIB.

La monitorización remota es una técnica mediante la cual, los datos son coleccionados por un agente sin intervención de una estación de gestión. RMON [Kooijman, 94] proporciona una buena aproximación a la gestión remota, sin embargo, todavía cuenta con muchas limitaciones, y no verifica los objetivos para los que fue creado, es decir, operación off-line, monitorización proactiva, datos de valor añadido, detección de problemas y aviso de los mismos. La utilización de agentes extensibles intenta solventar las deficiencias que presenta el RMON tradicional.

Los agentes RMON permiten al gestor de red asignar dinámicamente tareas y enviar programas a un determinado elemento de gestión. Un agente extensible infiere la funcionalidad de un agente RMON existente incluyendo, a su vez, una MIB extensible y una maquina virtual. La MIB extensible almacena los programas e información de control asociada, mientras que la máquina virtual es la encargada de ejecutar los programas almacenados en la MIB.

Los agentes extensibles proporcionan un mecanismo a través del cual, el gestor de red puede construir aplicaciones personalizadas dentro del marco de trabajo de RMON. Los gestores pueden escribir programas en un lenguaje interpretado que son almacenados y distribuidos como cualquier otro dato RMON. Los programas pueden ser inicializados manualmente por el sistema de gestión de red o automáticamente mediante el grupo de alarmas de RMON, con esto se consigue la operación off-line independiente del gestor. La escalabilidad se aumenta también dado que el sistema de gestión no necesita obtener información acerca del estado del programa extensible. Además el posible cuello de botella del modelo centralizado (gestor) es aliviado ya que los datos son coleccionados y generados localmente por cada agente extensible.

Las ventajas de utilizar un agente extensible son, fundamentalmente:

- Programación dinámica: La funcionalidad del gestor no está limitada por diseños previos de la MIB. La funcionalidad extra, es incorporada dinámicamente mediante los programas extensibles.

- Reducción de la sobrecarga de la red: El procesamiento de los datos recolectados por el agente RMON puede ser realizado por el agente extensible, sin necesidad de intervención del propio gestor. Además los datos procesados suelen tener menor tamaño que la propia información recolectada por lo que el trasiego de información hacia el gestor.
- Aumento de la inteligencia: Los agentes extensibles son un buen método para la distribución de la gestión en el sentido de que el procesamiento real de los datos no se realiza en el gestor.
- Localización de gestión de fallos: El procesamiento local de los datos generados por la RMON MIB añade flexibilidad a la detección de fallos. Cuando se produce un fallo, puede activarse una acción correctiva automáticamente. El nivel de tolerancia a fallos se incrementa pues el gestor trabaja únicamente con datos pre-procesados a de valor añadido.

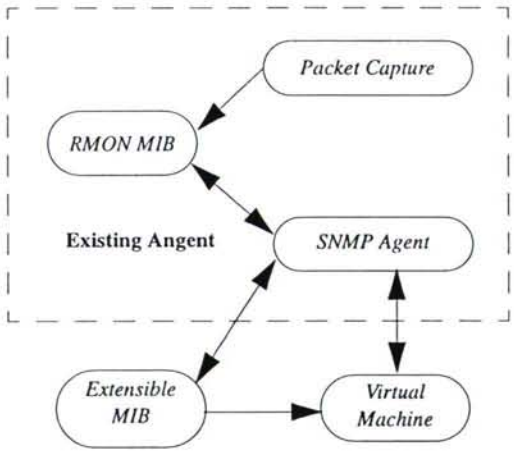

Figura 2.19. *Componentes de un agente extensible.*

La figura 2.19 muestra los componentes de un agente extensible. Se ha separado la RMON MIB de la MIB extensible dado que, por un lado, se utiliza distinto protocolo de acceso (SNMP para la RMON MIB y CMIP para la MIB extensible) y además la RMON MIB resta transparencia de localización.

La MIB extensible almacena la información de cada programa que va a ser ejecutado en al maquina virtual, consta de tres grupos:

- El *Control Group* que es utilizado para controlar la ejecución de los agentes extensibles. La figura 2.20 muestra la los posibles valores del campo *Control Status* así como su interrelación.
- El *Program Group* que almacena el código del programa que va a ser pasado a la máquina virtual.

- El *Data Group* que almacena las variables que han de estar disponibles para el sistema de gestión.

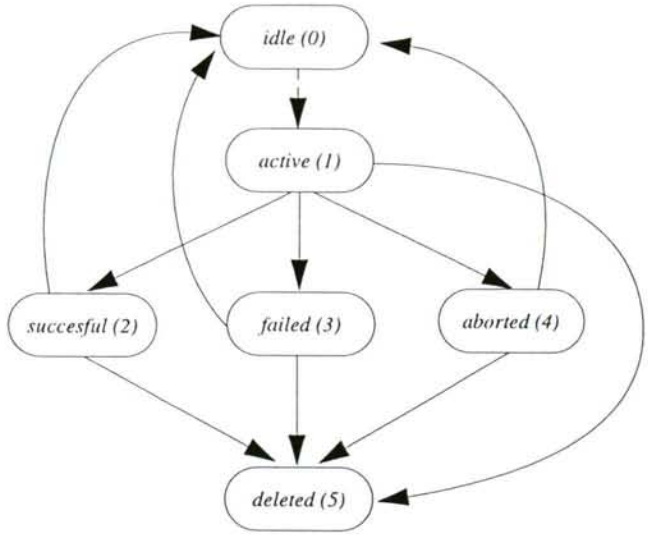

Figura 2.20. *Máquina de estados del agente extensible.*

Esta aproximación utiliza el protocolo estándar CMIP para almacenar el código extensible en la RMON MIB, por lo que resulta fácil de configurar y manejar. También muestra una gran flexibilidad de expansión, al poder incorporar nuevo código de manera sencilla a los agentes que en ese momento están siendo ejecutados.

La solución también reduce la sobrecarga de red, debido a que el procesamiento que realizan los agentes extensibles evitan que muchos de estos datos circulen innecesariamente por la red. El sistema es también escalable con facilidad, pudiendo añadirse muchos agentes con su maquinas virtuales correspondientes.

En cuanto a la seguridad, el uso de SNMPv1 como protocolo para el transporte de información de gestión hace que se hereden todos los inconvenientes que ya han sido citados anteriormente. Además la información almacenada en las MIBs carece de riqueza semántica y no pueden utilizarse estructuras de datos complejas para modelar los elementos de red.

Cuando ocurren fallos en la red o en uno de los gestores, pueden invocarse agentes extensibles que realicen las labores de gestión perdidas. De este modo, el sistema se convierte en tolerante a fallos.

Otro de los problemas asociados a esta arquitectura es la ausencia de mecanismos que permitan integrar nuevos protocolos de gestión de red y acceder a aplicaciones previamente desarrolladas, al menos de forma sencilla.

2.4.2 Agentes distribuidos basados en scripts

Algunas aproximaciones a la gestión de red distribuida están basadas en la ejecución remota de *scripts*. Muchos son los ejemplos que podemos encontrar en la literatura [Case, 93c; Gray, 95;

Ohta, 97; Schon, 95] acerca de este tipo de aproximaciones, aunque todas presenta características similares.

En [Kooijman, 95] se propone un método de gestión que extiende la carga de trabajo de los sistemas de gestión de red mediante el uso de agentes distribuidos. Con pre-procesado de datos llevado a cabo por estos agentes, pueden conseguirse reducciones en la sobrecarga de la estación de gestión y disminución del ancho de banda consumido.

Esta propuesta cuenta con un sistema central de almacenamiento de información de gestión, pero parte de los trabajos de gestión es delegado a los agentes mediante *scripts*. Cada uno de estos agentes gestiona su propia área de gestión local. Puede ser configurado para coleccionar y agregar todos los datos que haya pactado con la estación central y los resultados son entregados al gestor de forma preprocesada.

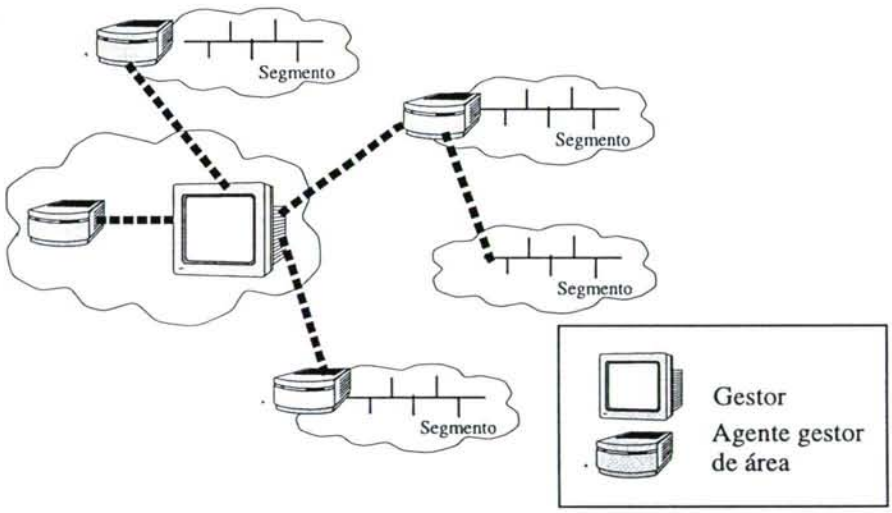

Figura 2.21. *Arquitectura de agentes gestores de áreas.*

La figura 2.21 muestra este modelo de gestión. Muchos de los segmentos cuentan con su propio agente gestor de área, aunque algunos agentes pueden ser configurados para gestionar más de un segmento. Las tareas a ejecutar por estos agentes no son fijas y pueden ser instanciadas por el gestor mediante la transferencia de *scripts* definidos por el usuario. De este modo podemos incorporar la gestión de nuevos protocolos, sin más que añadir los *scripts* adecuados. Estos *scripts* pueden ser programados en cualquier lenguaje de programación mediante la incorporación de unas extensiones. Los agentes de área puede comenzar con un conjunto predefinido de *scripts*, de los cuales múltiples gestores pueden ejecutar instancias personalizadas de los mismos. Un gestor puede transferir un nuevo *script* al agente cada vez que necesite mayor funcionalidad en el mismo. El control de *scripts* es llevado a cabo mediante una MIB especial que permite a un *script* ser cargado en un agente, iniciado, suspendido, configurado o eliminado.

Los *scripts* de un agente de área sólo son ejecutados si ocurren ciertos eventos de interés para el *script*, mientras tanto permanecen en un periodo de espera. Existen dos tipos de eventos que pueden generarse: (1) eventos generados por causas externas y (2) eventos generados dentro del sistema del agente.

Un *script* puede registrar funciones para el control de la gestión del sistema:

- *init*: Esta función es invocada automáticamente cada vez que el *script* esta siendo inicializado. Se utiliza para que el *script* coloque en la MIB sus parámetros de invocación y de resultado.
- *start*: Invocada cuando el *script* comienza su ejecución. Se utiliza para registrar funciones que den servicio a eventos locales.
- *stop*: Cuando se genera este evento se suspende la ejecución del *script*. Es decir, el *script* no responde a eventos hasta que recibe uno de liberación (resume). Esta función guarda el estado y los eventos registrados.
- *resume*: Notifica al *script* que se han completado todos los eventos. El *script* reinicializa el gestor de eventos, restaura su estado, etc.
- *exit*: Este evento se genera automáticamente cuando el gestor elimina una ejecución de un *script*.

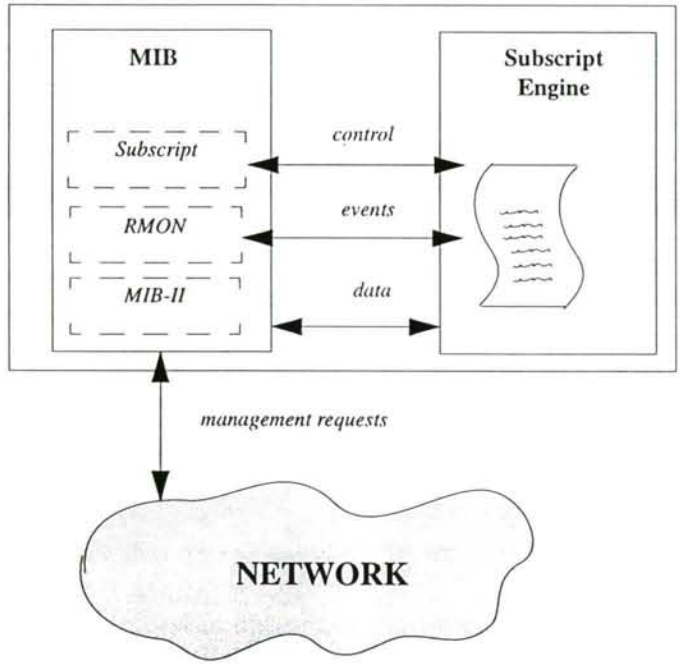

Figura 2.22. *Arquitectura de un agente de área.*

Un *script* puede registrar funciones adicionales para eventos cuyo origen es el propio agente.

- *timer*: Funciones registradas para ser invocadas en intervalos específicos un número determinado de veces.
- *packet*: funciones que pueden ser llamadas cuando los paquetes llegan a una determinada interfaz de red.
- *filter*: Similar al evento *packet* pero la función registrada es invocada cuando los paquetes recibidos casan con ciertos criterios.
- *channel*: Cuando un canal casa un paquete, cada función registrada es invocada para entregar el paquete. Así mismo, estas funciones actúan como recolectores de paquetes.
- *event*: La función es invocada cuando se emiten un cierto número de eventos del mismo tipo.

- *mib*: La función es invocada cuando se realiza una petición SNMP sobre alguna variable de la MIB.

Para definir los ficheros fuente y los argumentos de los *scripts*, se ha definido una MIB especial denominada *Subscript* MIB que consta de dos grupos de variables: *Source Group* y el *Run Group*. El *Source Group* debe ser usado para cargar los ficheros fuente del *script* en el agente y prepararlo para su ejecución. El *Run Group* consiste en múltiples instancias del mismo código fuente aunque, posiblemente, con diferentes argumentos de inicialización. De este modo, varias instancias de un mismo *script* pueden ser ejecutadas simultáneamente, cada una con su propio contexto.

El uso de *scripts* que se pueden ejecutar de manera remota en un agente tiene connotaciones de seguridad e integridad derivadas de las posibles responsabilidades de accesos por usuarios sin permisos. Con los agentes por área esto puede ser solucionado no permitiendo la ejecución de comandos peligrosos, cada comando de este tipo debe ser colocado en un lugar especial en el que sólo se permitirá su ejecución si cuenta con los permisos adecuados.

Esta aproximación utiliza el modelo de datos SNMP con lo que incorpora toda la problemática asociada: falta de riqueza semántica, limitado acceso a la información de tipo masivo, falta de distribución de la información, etc.

El sistema presenta ventajas en cuanto a fiabilidad y autonomía de los agentes, dado que puede responder a eventos que desencadenan la ejecución de *scripts* en respuesta a ese fallo.

Otro de los inconvenientes que presenta la extensión de agentes mediante el uso de *scripts* es que cuando un gestor inicia un *script* en un agente, pierde el control sobre la ejecución del mismo. Es posible que durante su ejecución, se produzcan excepciones que lo bloqueen y que impidan al gestor terminar su ejecución para activar otras tareas.

2.4.3 Modelo de agentes activos (AMOs)

En [Vassila, 95] se critica el modelo clásico agente-gestor y se propone una solución más flexible mediante la introducción de los *Active Managed Objects* (AMOs) que son entidades remotas programables capaces de ejecutar procedimientos de gestión externos localmente. Estas tareas son definidas en términos de lenguajes *script*. El modelo se basa en la definición de objetos especiales, definidos en la MIB del sistema y que puede ser accedidos por aplicaciones de gestión como cualquier otro objeto.

Los AMOs una vez definidos, pueden ser creados, borrados, los procedimientos de gestión modificados, con lo que proporcionan una buen mecanismo para especificar comportamientos en sistemas distribuidos. Mediante este modelo puede fácilmente modificarse los objetivos de las políticas de gestión. En la figura 2.24 se muestra como puede ser ejecutado una aplicación de gestión en forma de *script* de manera local en un elemento de red. Esta forma de ejecución contrasta con la manera tradicional que puede observarse en la figura 2.23.

La estructura interna de un AMO facilita el encapsulamiento necesario para interpretar y ejecutar procedimientos de gestión. La parte activa de los AMOs es la que facilita la comunicación con el compilador/intérprete y toma las acciones oportunas en base a la información que recibe. Las principales tareas de un AMO son:

- Aplicar métodos de control de acceso a aplicaciones de gestión que pretendan realizar procedimientos de gestión. Es el encargado, por tanto, de garantizar los derechos de acceso de ciertos gestores, rechazando accesos no permitidos.

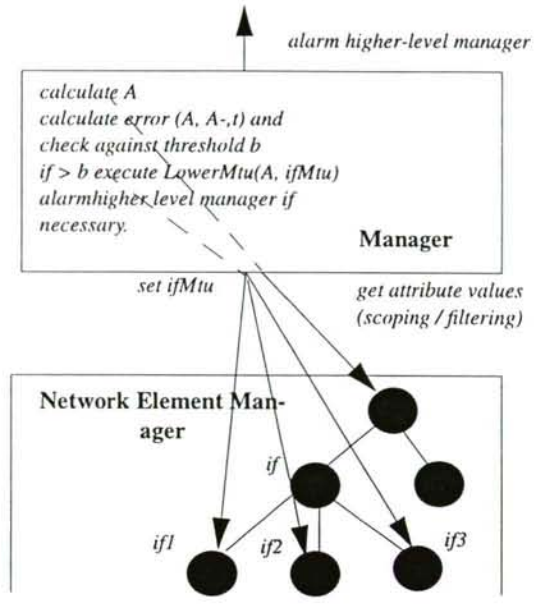

Figura 2.23. *Ejecución tradicional de una tarea de gestión.*

- Permitir y planificar la ejecución repetida de procedimientos de gestión. Los AMOS no necesitan ser monitorizados sino que se comunican con el gestor de manera asíncrona, únicamente cuando ocurre algún evento.
- Chequear la integridad y corrección de los procedimientos de gestión. Debe facilitar mecanismos que rechacen procedimientos que contienen comandos que pueden resultar peligrosos para el sistema.
- Presentar los resultados a la aplicación de gestión.
- Facilidades de almacenamiento de procedimientos de gestión ante fallos para evitar la carga de nuevo en caso de condiciones anómalas de fallo.

Es muy importante el uso de un lenguaje de gestión apropiado. En este caso se ha seguido la política de utilizar uno ya existente y no crear uno nuevo para esta aplicación específica. A la hora de abordar la elección del mismo se han seguido los siguientes aspectos:

- Debe tener un tamaño pequeño de modo que su ejecución por parte de un AMO no tenga un impacto muy grande en los gestores de elementos.
- Debe ser de alto nivel y simple, pero que al mismo tiempo pueda expresar fácilmente las operaciones de gestión. Estar orientado hacia los requerimientos del gestor humano y soportar esquemas de gestión triviales y sofisticados.
- No debe tener puntos peligrosos, en el sentido de poder llevar a cabo operaciones que pueden dañar el sistema.
- Debe facilitar la ejecución concurrente de programas de gestión.

- El lenguaje debe proporcionar un adecuado nivel de abstracción para facilitar las operaciones de gestión con buenos mecanismos de control, seguridad y acceso que puedan ser encapsulados en comandos y funciones.

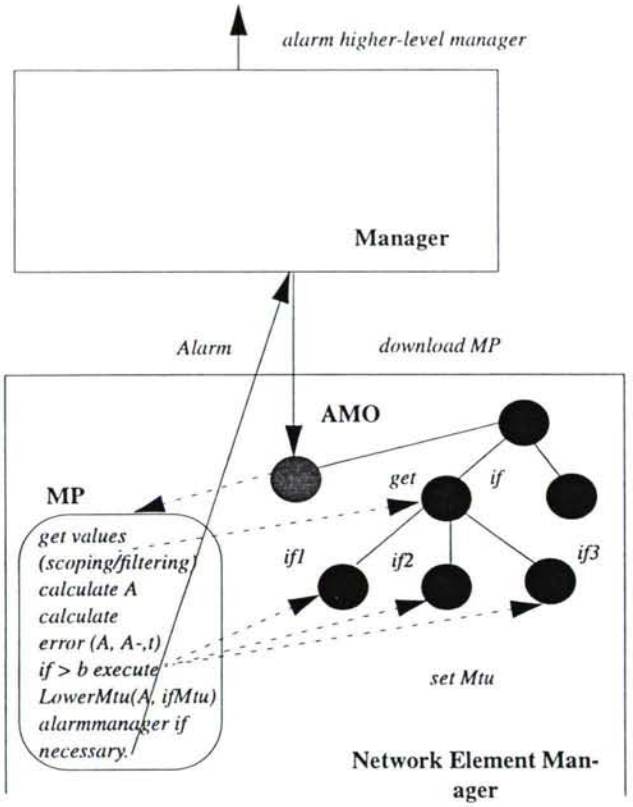

Figura 2.24. *Tarea de gestión utilizando AMOs.*

Se ha implementado un proyecto piloto de AMOs que actúa como una aplicación de OSIMIS (citada anteriormente). Los AMOs cooperan con el Coordinador en la ejecución de los procedimientos de gestión. Una de las facilidades que se consigue con esta inclusión es la posibilidad de concurrencia dentro de los agentes OSIMIS. Como lenguaje de gestión se ha utilizado el CMIS-Tcl que extiende el intérprete de Tcl [Ousterhout, 94] para facilitar el uso de funciones CMIS.

Sin embargo este modelo presenta todavía algunos problemas:

- El modelo de ejecución de AMOs juega un papel muy importante en la funcionalidad del sistema. Sin embargo, existe una separación entre la invocación de un *script*, que es controlada por el agente y la propia ejecución del mismo, controlada y dirigida por el propio *script*.
- El uso de un lenguaje específico restringe mucho la funcionalidad y flexibilidad del gestor, al tener que adaptarse a ese lenguaje.
- Los AMOs que residen en el mismo o diferente sistema de gestión necesitan complejos sistemas de comunicación.
- Deben tenerse en cuenta los problemas que puede ocasionar la terminación anormal de *scripts* para la aplicación agente.

- La necesidad de construir una arquitectura superior para soportar las funcionalidades de los AMOs. Debe pensarse en aplicaciones tipo CORBA para implementar este tipo de arquitecturas dado que hace transparente el acceso a métodos y esquemas de nombrado.

2.4.4 Modelo DEAL

En [Znaty, 95] se propone un modelo de gestión basado en delegación mediante un lenguaje de *scripts*. Este lenguaje se denomina DEAL y proporciona una sintaxis de alto nivel para el desarrollo rápido de funciones de gestión. La delegación se introduce para minimizar tanto el uso de los recursos de la red como el tiempo de procesamiento por parte del gestor.

El protocolo SNMP puede ser visto como un lenguaje de bajo nivel, parecido al ensamblador que no facilita el desarrollo de aplicaciones de gestión. Las funciones de alto nivel requieren otro tipo de procedimientos como es el caso del filtrado de datos. Con el uso de DEAL se pretende cubrir dos objetivos:

- Minimizar el tiempo que el desarrollador tiene para construir las aplicaciones de gestión. DEAL es un lenguaje interpretado basado en peticiones SQL (*Structured Query Language*).
- Minimizar el uso de los recursos y el tiempo de procesamiento por parte del gestor al introducir delegación, consistente en enviar los scripts en DEAL a los agentes de gestión remotos, enlazando estos dentro del entorno local y ejecutándose bajo control local o remoto. Previamente, los scripts DEAL son traducidos a scripts SNMP, con lo que no es necesario realizar grandes cambios en los agentes SNMP actuales

El lenguaje propuesto consiste en cuatro operaciones básicas: SELECT, UPDATE, GET y SET. SELECT y UPDATE son utilizadas como peticiones SQL que permiten al gestor leer y actualizar tablas en la MIB bajo condiciones de filtrado. GET y SET son operaciones que permiten leer y modificar los valores de los atributos de la MIB SNMP. El uso de SQL está motivado, en gran medida, por el hecho de que los actuales servicios de gestión son desarrollados generalmente mediante lenguajes de bases de datos. Además es necesaria interacción entre diversas aplicaciones de gestión que pueden encargarse de áreas distintas, la utilización de SQL puede facilitar esta tarea.

El protocolo de la familia OSI, CMIP, facilita también las operaciones de *scoping* y *filtering*, sin embargo, mapear o trasladar peticiones CMIP a peticiones SNMP no es tan sencillo como hacerlo de SQL a SNMP dato que los modelos de datos son diferentes. CMIP es un protocolo orientado a objetos, mientras que SNMP es un protocolo orientado a atributos.

Se ha desarrollado un *parser* que permite trasladar scripts en DEAL a scripts en SNMP para aprovechar de este modo la funcionalidad de los paquetes de gestión que normalmente vienen con ciertos equipos de comunicaciones. La figura 2.25 describe este proceso. De este modo el script

SNMP puede ser ejecutado por un intérprete SCOTTY [Schon;'95], que es un intérprete TCL con algunas extensiones para SNMP.

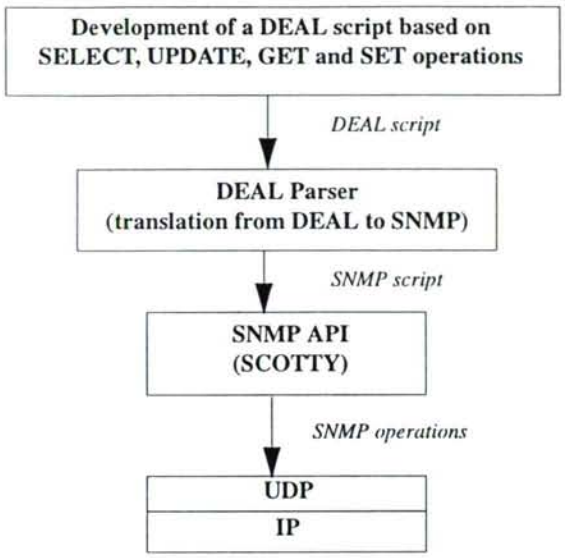

Figura 2.25. *Traslado de scripts DEAL a SNMP.*

Para el proceso de delegación se han incorporado unos gestores intermedios en la organización centralizada de SNMP, es la denominada gestor de gestores. Este gestor intermedio es el encargado de:

- Ejecutar los scripts DEAL que han sido enviados por el gestor.
- Realizar complejas operaciones de filtrado para excluir cualquier información que no sea de interés para el gestor.
- Agregar información para proporcionar una mejor visión global al gestor de nivel superior.

La figura 2.26 muestra también como los gestores pueden llevar a cabo llamadas sobre los agentes delegados que están instalados en la misma máquina que el agente SNMP.

En esta configuración podemos indicar que:

- El agente delegado y el agente SNMP se encuentran en la misma localización, lo que reduce el número de recursos necesarios.
- Los gestores se encuentran generalmente en otras máquinas distintas.
- El agente delegado realiza peticiones SNMP sobre el agente SNMP.
- El agente SNMP puede enviar *traps* al gestor.

- El resultado de una petición del agente delegado que devuelve al gestor es formateado como una tabla SQL si el script DEAL consiste en una petición de SELECT.

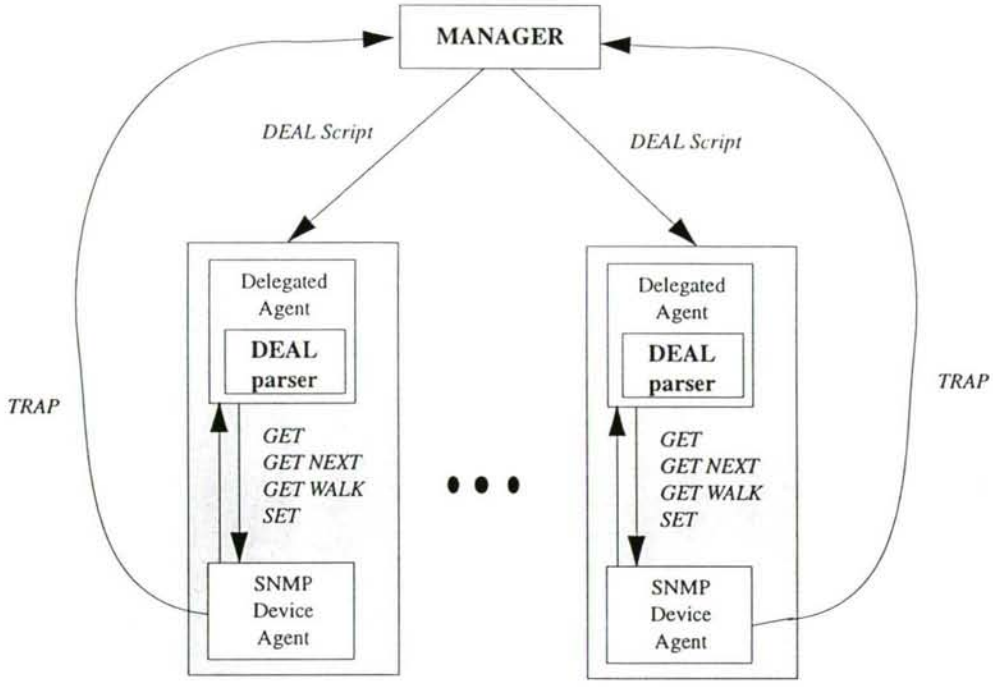

Figura 2.26. *Organización propuesta.*

La eficiencia de este método depende enormemente de los siguientes parámetros:

- La carga de tráfico de la red.
- La distancia entre el gestor y el agente
- El número de equipos que están siendo gestionados por el mismo gestor.
- La complejidad de las tareas de gestión.
- La proporción de filas relevantes si el gestor realiza peticiones SELECT o UPDATE.

2.4.5 Gestión por delegación

Un ejemplo de sistema de gestión distribuido es el presentado en [Gold, 94; Gold, 95b], donde se propone un paradigma distribuido de gestión, en el cual todo el control e inteligencia se encuentra distribuido por las distintas entidades de red. Este modelo, denominado gestión por delegación (MbD), utiliza un paradigma distribuido con la ventaja del incremento de la potencia computacional de los agentes de red y la mejora del ancho de banda necesario para llevar a cabo gestión de red. La MbD soporta tanto distribución temporal (realizar varias funciones al mismo tiempo) como espacial (realizar esas funciones en distintos dispositivos de red). En este modelo, los agentes son capaces de llevar a cabo sofisticadas funciones de gestión de manera autónoma, descargando al gestor o gestores de estas tareas y reduciendo la sobrecarga que se produce en la red por el intercambio de mensajes de gestión. Sigue utilizando un protocolo de gestión de red para comunicarse con el gestor, pero consigue la distribución mediante el incremento de la autonomía de gestión de los agentes.

En este modelo se define un tipo de proceso distribuido (*Elastic Process*) [Gold, 93] que soporta extensión del tiempo de ejecución y contracción de la funcionalidad. Es decir, durante su ejecución un proceso de este tipo puede absorber nuevas funcionalidades delegadas por otros procesos. Estas funciones pueden ser invocadas por clientes remotos que tengan acceso a este tipo de procedimientos remotos. De este modo, la MbD proporciona una eficiente y escalable gestión de sistemas mediante el uso de agentes eslásticos. En vez de mover datos de los agentes al gestor en donde se encuentra la aplicación que los procesa, el modelo MbD mueve la aplicación a los agentes que contienen los datos necesarios delegando un proceso elástico.

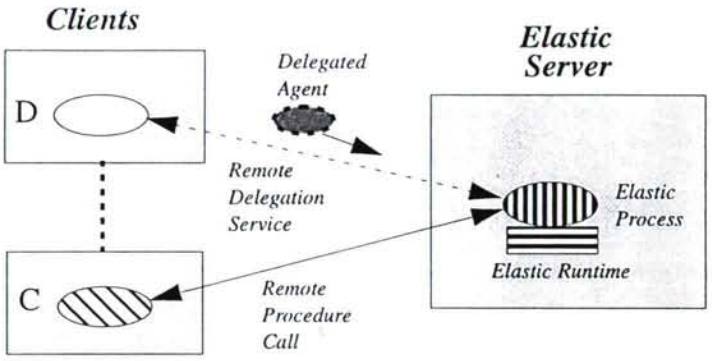

Figura 2.27. *Delegación de agentes móviles en un servidor elástico.*

Este tipo de sistemas tiene sentido para aquellos tipos de aplicaciones de gestión que requieren o pueden hacer uso de las ventajas de la distribución espacial para, por ejemplo, reducir la sobrecarga de red o los retardos. Existe una clase de aplicaciones de gestión, especialmente las que evalúan y reaccionan a eventos, que deben ser distribuidas en los agentes y no pueden ser procesadas eficientemente por un gestor central. La facilidad para descargar funciones en los agentes cuando la red está sobrecargada facilita enormemente el acceso a las mismas cuando la red está sobrecargada, reduciendo el ancho de banda que consume una aproximación centralizada.

Estos agentes vienen a solucionar el problema de los actuales modelos clientes/servidor que se caracterizan por su rigidez, dado que un cliente solo puede acceder a una serie de servicios predefinidos por parte del servidor.

Un servidor elástico es un proceso elástico que presenta una interfaz dinámica, una colección de servicios, que pueden ser invocados remotamente por sus clientes. Además alguno de esos procedimientos o servicios puede a su vez ser delegado.

La delegación consigue un incremento significativo del rendimiento general del sistema dado que decrementa enormemente el ancho de banda y la latencia para aplicaciones distribuidas, que es el principal problema con el que se encuentran los tradicionales sistemas cliente/servidor, dado que si bien la potencia de los procesadores se incrementa año tras año de manera significativa no lo hace así la velocidad de la red, con lo que ésta introduce un gran cuello de botella

La Gestión por Delegación incrementa la gestionabilidad para la distribución dinámica de inteligencia a dispositivos de red, mediante los agentes extensibles. Los procesos de gestión

pueden delegar a los agentes MbD la ejecución de programas de gestión que monitorizan y controlan los objetos locales de manera efectiva sin necesidad de utilizar gestores remotos.

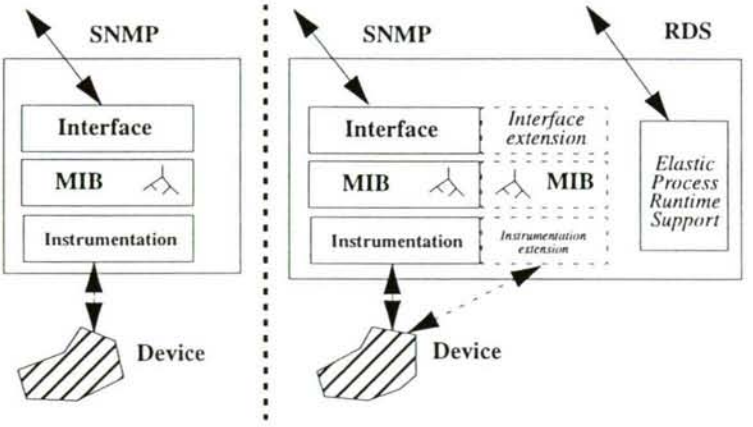

Figura 2.28. *Extensión de un agente SNMP.*

Los agentes MbD pueden interoperar con protocolos de red como es el caso de SNMP. De este modo, pueden beneficiarse de las aplicaciones que han sido desarrolladas para estos protocolos estandarizados. La figura 2.28 (a) muestra una agente SNMP que incluye un interfaz SNMP, una MIB y la correspondiente instrumentación para obtener los datos del dispositivo. La figura 2.28 (b) muestra como la utilización de un agente MbD puede incrementar la funcionalidad de los servicios de interfaz, MIB e instrumentación. Por ejemplo, el gestor de red puede delegar un procedimiento que implementa funciones de filtrado sobre la MIB SNMP. Este proceso local, almacena los valores filtrados en una tabla de la MIB que puede ser recuperada por el propio gestor. Esta rutina puede ser invocada por otros gestores o clientes.

Además, el uso de agente MbD facilita la adquisición, por parte de los dispositivos, de autonomía de gestión. Por ejemplo, cuando se pierde la comunicación gestor-agente los dispositivos pueden activar programas que proporcionen una serie de instrucciones de gestión predefinidas para la operación autónoma mientras no se restaure la conexión.

En la literatura, podemos encontrar otros modelos de delegación diferentes al planteado en el paradigma de MbD. La evaluación remota [Stamos, 90] permite que las expresiones de programas sean transferidas desde una computadora cliente a una computadora servidor. El servidor la evalúa y devuelve el resultado al cliente. De este modo solvente el problema de los servidores rígidos, dado que no es necesario establecer todos los procedimientos en tiempo de diseño. Sin embargo, la ejecución remota de los programas a evaluar es síncrona y el cliente no tiene control sobre la ejecución del programa en el servidor. Por ejemplo, no puede cancelar su ejecución cuando se produce una ejecución demasiado larga. Además el intercambio de programas sólo puede realizarse entre computadoras con la misma arquitectura y que utilicen el mismo lenguaje.

Otro modelo, conocido como *Remote Programming* [White, 94] requiere un programa especializado, *Telescript*, para construir agentes que puedan ser transferidos a interpretes remotos para su ejecución. Sin embargo, la necesidad de una comunicación fiable entre interpretes hace inviable esta solución en ciertos entornos de red.

2.4.6 Modelo basado en dominios de gestión

Los proyectos ESPRIT SYSMAN (7026) y IDSM (6311) especifican una arquitectura para la gestión distribuida de sistemas basada en el concepto de dominios [Sloman, 89]. Un dominio es un grupo de objetos que permite particionar el sistema de gestión, con vistas a crear un gran número de sistemas distribuidos interrelacionados.

En [Sloman, 93] se propone una arquitectura para la gestión de sistemas distribuidos basada en el concepto de dominios. Como característica fundamental de este modelo está el uso de un servicio de políticas que sirve de soporte para la especificación, almacenamiento y manipulación de directrices que pueden ser usadas para modificar el comportamiento de los gestores. La arquitectura hace uso de las técnicas de procesamiento distribuido basado en objetos y extiende los conceptos de OSF DME [OSF, 92].

La separación de las políticas de los componentes del gestor, ver figura 2.29, facilita que los gestores puedan ser reutilizados en diversos entornos, proporcionando políticas específicas a cada uno de ellos. Por otro lado el número de objetos a gestionar en un entorno de red es muy elevado por lo que resulta inviable especificar una política individual para cada uno de ellos, es necesario, agrupar estos objetos en lo que se ha dado en denominar dominios de gestión. En este modelo de gestión se representan por objetos tanto los recursos a gestionar como los gestores [Moffett, 93].

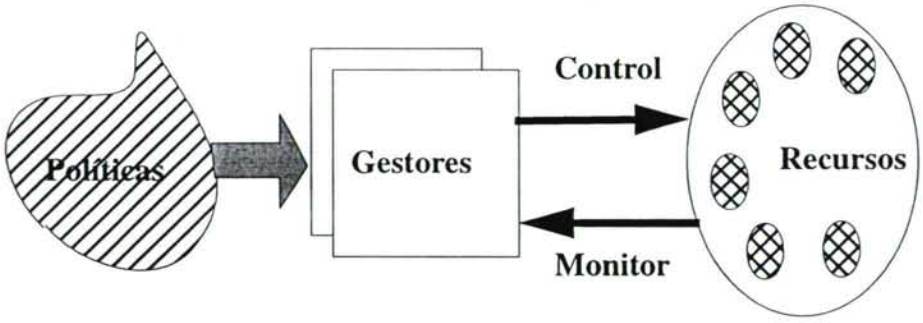

Figura 2.29. *Gestion basada en políticas.*

Cada uno de los objetos a gestionar presenta una o más interfaces, ver figura 2.30,que representan diferentes vistas de gestión del recurso que está modelando, de este modo pueden existir diferentes interfaces para la gestión de la seguridad, gestión de fallos, gestión de rendimiento, etc. Además un recurso puede ser representado por más de un objeto según el propósito de gestión que se persiga.

El gestor es el responsable de un conjunto de agentes. Puede cooperar con otros gestores mediante una filosofía jerárquica o mediante protocolos entre entidades pares. Se define un gestor *proxy* como un objeto gestor que se coloca en el medio de otro gestor que está realizando la gestión. Estos objetos también cuentan con una interfaz de gestión. A su vez, un gestor puede actuar como gestor de otro gestor. Del mismo modo que en SNMP, se permiten tres tipos de interacciones, dos iniciadas por el gestor, como la petición de datos y el establecimiento de valores, y una iniciada

por el agente, las notificaciones. En ciertas plataformas, esta interfaz tiene que ser implementada mediante una técnica de mensajes y no directamente.

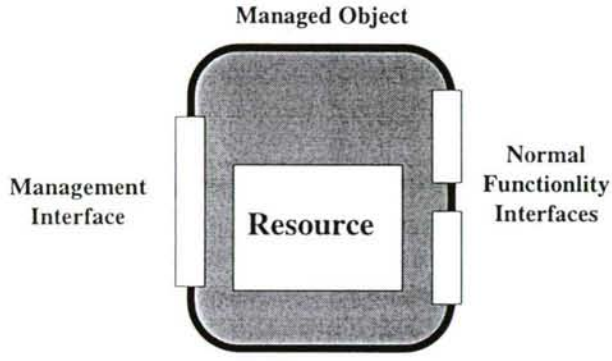

Figura 2.30. *Objeto de gestión con recursos encapsulados.*

También se permite definir un objeto compuesto, ver figura 2.31, que puede incluir a uno o varios gestores responsables de objetos gestionados. El objeto compuesto es considerado un objeto simple de modo que se encapsula su funcionamiento interior y se presenta hacia fuera con interfaces de gestión estándar.

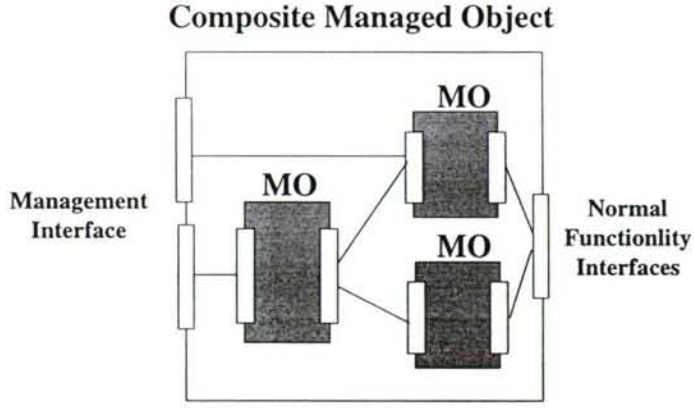

Figura 2.31. *Estructura de un objeto compuesto.*

La gestión de un sistema de información distribuida debe ser estructurada y delimitar la responsabilidad entre los distintos gestores. El concepto de dominio proporciona el marco de trabajo para particionar la responsabilidad de gestión, agrupando objetos con la finalidad de especificar políticas comunes. Un dominio es un objeto que mantiene una lista de referencias acerca de los objetos miembros que está gestionando. Si un dominio hace referencia a un objeto, este es visto como un miembro directo del dominio y el dominio es visto como su padre. A su vez un dominio puede formar parte de otro dominio, de este modo se forman los denominados subdominios, de este modo pueden aplicarse también políticas específicas a grupos de objetos. Los miembros de un subdominio son miembros indirectos del dominio padre. De esta forma, un objeto puede ser un miembro directo o indirecto de múltiples dominios. Pueden existir dominios superpuestos, que son aquellos que tienen uno o más objetos en común. Un dominio no encapsula a los objetos que lo componen, esto se lleva a cabo con los objetos compuestos.

La gestión de sistemas distribuidos involucra la monitorización de la actividad de un sistema, tomando decisiones de gestión y realizando acciones de control con vistas a modificar el comportamiento del mismo. Se ha definido el medio mediante el cual los gestores pueden monitorizar y controlar el comportamiento de los objetos e invocar operaciones de gestión basándose en esto. Las políticas de gestión rigen el proceso de toma de decisiones. Una política es información que influye en el comportamiento del objeto, es decir, la información que influye en las interacciones entre un objeto y un sujeto. Los dominios son utilizados para definir políticas comunes a varios objetos. Se pueden aplicar múltiples políticas sobre un objeto y a su vez, un objeto puede ser sujeto/objeto de múltiples políticas. Se definen tres tipos de políticas:

- *Políticas de gestión*: Especifica la interrelación entre un conjunto de gestores y un conjunto de objetos gestionados, especificando que gestores están obligados o autorizados a realizar actividades en los objetos de gestionados.
- *Políticas de autorización*: Define que actividades de un gestor (operaciones) pueden llevarse a cabo sobre un determinado dominio de gestión. El uso de este tipo de políticas puede desencadenar conflictos que deben resolverse mediante reglas de precedencia.
- *Políticas de obligación*: Define las actividades que un gestor debe o no llevar a cabo.

Las políticas [Moffett, 91] pueden ser vistas como un objeto no activo, lo que representa una ventaja considerable dado que pueden agruparse en dominios y de este modo, establecer políticas sobre un conjunto de políticas.

Como muestra la figura 2.32, un sistema de gestión distribuida consta de un conjunto de aplicaciones que tienen los correspondientes interfaces de usuario. Estos hacen uso de un conjunto común de servicios de gestión para monitorizar y manipular tanto las políticas como los dominios. Los objetos de gestión pueden interactuar usando varios servicios de comunicación para cumplir los requerimientos demandados por una aplicación particular. Aunque la figura 2.32 presenta una arquitectura jerarquizada, esto no es necesariamente así, dado que una aplicación de gestión puede necesitar varios servicios de configuración, seguridad, etc.

De este modo, cada aplicación de gestión es construida mediante un conjunto de componentes cooperativos. Se establecen los siguientes conjuntos de servicios de gestión:

- Servicios de dominios y políticas que permiten crear, eliminar, insertar objetos, listarlos, etc. Se dispone de una función de distribución de políticas que utiliza estos servicios para determinar que políticas se aplican a determinados dominios.
- Servicios de procesamiento distribuido como puede ser el soporte de transacciones atómicas, sincronización de procesos, directorios de servicios, etc.
- Servicios de objetos distribuidos: Permite la distribución transparente del sistema de gestión, facilitando la cooperación entre gestores y objetos gestionados sobre un gran número de nodos gestionados. Debe permitir la configuración en términos de creación, borrado y posible migración de objetos entre nodos. También debe permitir la localización transparente de invocaciones remotas y notificaciones de eventos.

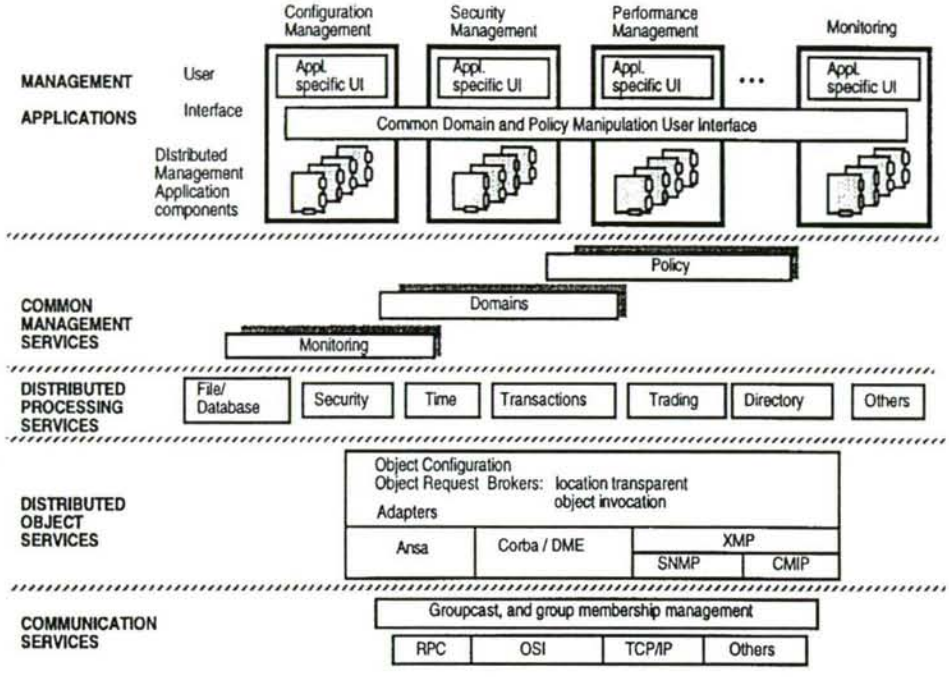

Figura 2.32. *Arquitectura del sistema de gestión distribuido.*

- Servicios de comunicación: Los gestores pueden hacer uso de variedad de diferentes servicios de comunicaciones. Es necesaria la utilización de servicios de llamadas remotas a procedimientos para invocación de operaciones y de un servicio de mensajes asíncronos para notificaciones. También se necesitan mensajes de grupo para invocar servicios a varios objetos simultáneamente.

2.4.7 Modelo DOMINO

En [Sloman, 94] se describe el proyecto DOMINO (*Domain Management for Open Systems*) que se basa en el uso de dominos, agrupaciones de objetos con distintas políticas de gestión. Durante el desarrollo de este proyecto, también se establecieron las base para crear un marco simple y flexible para la implementación de vistas de objetos. El proyecto DOMINO pretende la gestión de múltiples redes interconectadas, facilitando estos servicios sobre un gran número de organizaciones. La estrategia centralizada, no es la adecuada para gestionar este tipo de redes, por lo que debe adoptarse un modelo distribuido con mecanismos de negociación entre gestores independientes que cooperan sin perder su autonomía. Pueden identificarse dos conjuntos de actividades genéricas en un gestor de un sistema:

- Realizar operaciones de gestión para monitorizar y controlar el comportamiento de los objetos relacionados con funciones de gestión.
- Crear, interpretar y monitorizar políticas. Estas políticas son diferentes para las diversas operaciones de gestión. Una política es especificada y aplicada sobre un conjunto de objetos, constituyendo un medio para influir sobre las operaciones a realizar sobre ese grupo de objetos.

Un dominio de gestión, es por tanto, un grupo de objetos sobre los que puede aplicarse una política concreta, en otras palabras, han sido agrupados explícitamente por razones de gestión. constituyendo el concepto clave del proyecto DOMINO. Dado que los dominios son a su vez objetos, pueden ser agrupados en nuevos dominios. Los dominios pueden ser utilizados, además, para estructurar el espacio de nombres de los objetos. Los dominios son una generalización de los conceptos de directorios estructurados en forma de árbol de los sistemas de ficheros.

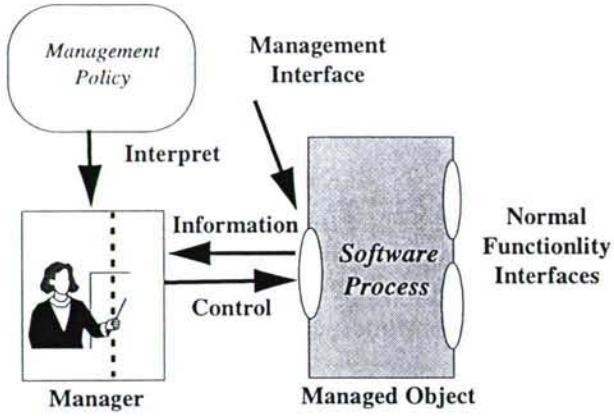

Figura 2.33. *Interacciones en el modelo DOMINO.*

No debe confundirse el concepto de agrupamiento con el de encapsulación, cuya interfaz exterior se comporta como un único objeto, Aunque este concepto es útil para la construcción de sistemas distribuidos, añade gran complejidad a su desarrollo y operación. Además a veces es necesario realizar operaciones únicamente sobre uno de los objetos agrupados, lo que la encapsulación no permite.

No es practico especificar a un miembro como perteneciente a un dominio en base a un predicado sobre alguno de sus atributos. Los miembros válidos de un dominio serán aquellos que han sido creados explícitamente para el mismo o han sido insertados.

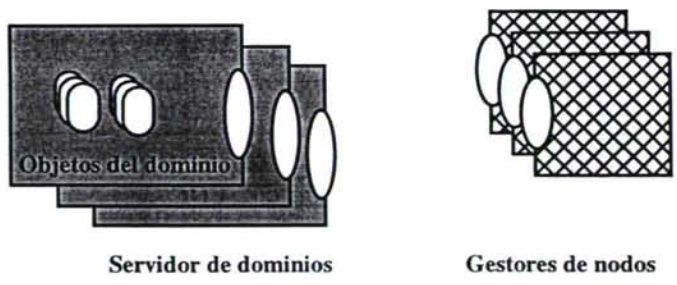

Figura 2.34. *Componentes del servicio de dominios.*

La implementación resultante es la mostrada en la figura 2.34. Existe un servidor de dominios que se encarga de almacenar y mantener los dominios correspondientes a un subárbol de la jerarquía de dominios. Si no dispone de la información relativa a una parte del subárbol cuenta con la dirección del servidor responsable de este subárbol. Esta implementación proporciona información particionada de dominios siendo posible la replicación de dominios con la finalidad de que aumente el rendimiento y disponibilidad. El otro aspecto importante es el gestor de nodos que reside en cada nodo físico que tiene objetos gestionados y mantiene información acerca de los

objetos locales así como de sus dominios padre. Actúa de agente local para interaccionar con el servidor de dominios y es responsable de mantener informado al servidor de dominios de posibles cambios acaecidos en los objetos involucrados. Además puede crear nuevos objetos bajo indicación del servidor de dominios. La ubicación de estos nuevos objetos en un determinado dominio puede traer consigo problemas de ciclos de pertenencia que deben ser solucionados manualmente.

Las reglas de acceso a esos dominios deben usarse para establecer un medio flexible de definición de interrelaciones entre dominios de gestores y objetos con vistas a definir que operaciones del gestor pueden llevarse a cabo sobre los dominios gestionados.

2.4.8 Modelos basados en CORBA

La *Common Object Request Broker Architecture* (CORBA) [Ohta, 97] es un modelo basado en la implementación de operaciones independientes e interfaces orientados a objetos. La figura 2.35 muestra la arquitectura general de este modelo.

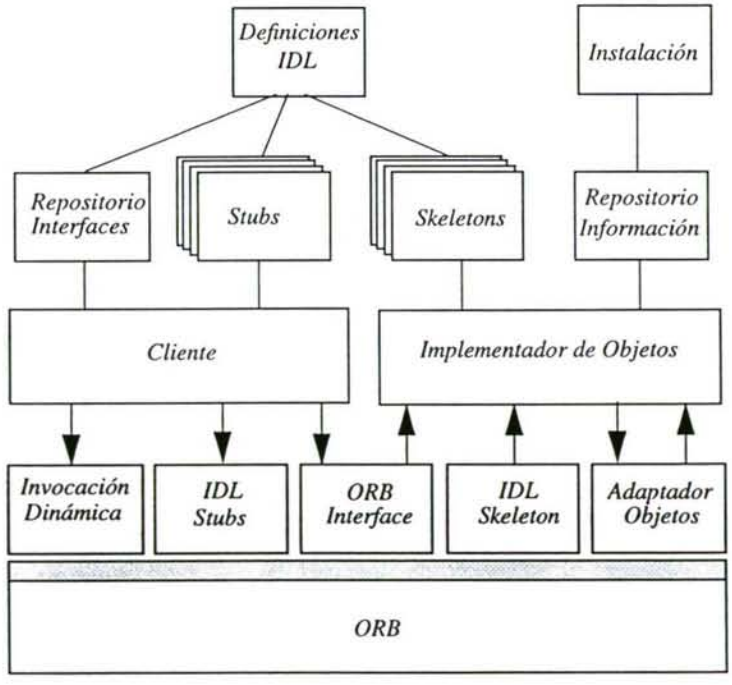

Figura 2.35. *Arquitectura CORBA.*

CORBA está organizado en agentes de peticiones de objetos (ORB) que permiten a las aplicaciones realizar peticiones a un objeto cualquiera de la red. Estas peticiones se establecen mediante interfaces definidas en un lenguaje de definición de interfaces (IDL). De este modo, la petición del cliente es independiente de la localización e implementación del método referenciado en el objeto solicitado. La función del ORB es encontrar la implementación del objeto referenciado, y comunicarle los datos de la petición.

El ORB es utilizado a través de varias interfaces por parte del cliente. Típicamente se utiliza el interface de invocación dinámica que permite a un cliente ser independiente del interface del objeto destino, y del mecanismo de invocación en el otro extremo. También se presentan interrelaciones entre *stubs* y *skeleton* de clientes y objetos. El repositorio de interfaces y el repositorio de

implementaciones están también disponible para que los clientes puedan hacer uso de ellos en sus invocaciones.

A su vez, un ORB puede llevar a cabo operaciones que involucren; (a) a operaciones sobre todas las implementaciones ORB, (b) operaciones sobre tipos de objetos determinados y (c) operaciones que son específicas a un estilo particular de implementación de objetos.

El lenguaje de definición de interfaces utiliza las mismas reglas que C++ pera más restrictivas. Utiliza los conceptos de tipos, herencia, atributos, nombres y *scoping* de manera parecida a como lo hace CMIP.

La utilización de este tipo de arquitecturas tendrá, sin duda, un gran peso en el desarrollo futuro de sistemas distribuidos. Sin embargo, actualmente existen otros sistemas para el propósito de la distribución de procesamiento como COM [Brock, 94], TINA [TINA, 95], ANSA [ANSA, 93] y para el propósito de la gestión de red como OSI [ITU-T X720] y SNMP. El objetivo de la mayoría de los modelos basados en CORBA es proporcionar un acceso transparente a esos sistemas ya maduros y muy extendidos.

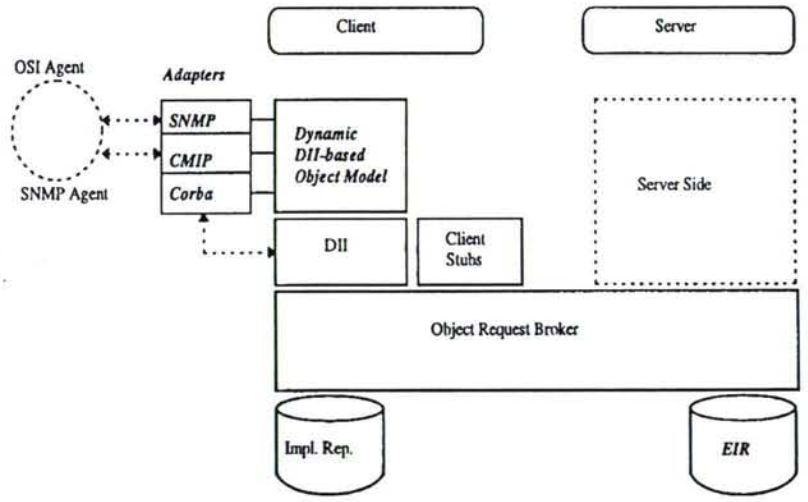

Figura 2.36. *Sistemas basados en CORBA.*

En [Ban, 96] se presenta una aproximación basada en un modelo de objetos genérico que usa CORBA como mecanismo de distribución y proporciona una API genérica a los clientes que pueden utilizar modelos estándares de gestión. Un componente fundamental de esta arquitectura es el concepto de adaptador (ver figura 2.36), que es un puente que permite traducir peticiones entre cualquiera de los sistemas involucrados. Estos adaptadores están colocados en la parte de cliente en vez de en el servidor como en el caso de los adaptadores CORBA. Un tercer componente de este tipo de sistemas es la extensión que se realiza sobre el repositorio de interfaces (IR) de CORBA para añadir meta-información acerca de los otros modelos involucrados. Todo esto va a posibilitar la creación de clientes con una interfaz de acceso abstracta e independiente de posibles cambios en los servidores. Además va a ser posible escribir clientes genéricos que no necesitan incluir en sus especificaciones toda su funcionalidad, sino que podrán cargarla dinámicamente del repositorio de interfaces extendido. Como resultado de todo esto, será posible la creación de gestores pequeños y con poco peso computacional.

Otras aproximaciones, como la introducida en [Schade, 96], trabajan sobre el servicio de eventos que proporciona CORBA para incorporar un repositorio de eventos que pueda ser utilizado por todos los objetos del sistema distribuido.

El uso de CORBA para el desarrollo de sistemas de gestión de red, se centra, fundamentalmente, en mecanismo de distribución para la integración de modelos de gestión ya existentes. En la literatura pueden encontrarse muchas alabanzas a este modelo, pero también son numerosas las críticas con respecto al rendimiento que ofrece el uso de este paradigma. CORBA es una herramienta muy potente para la distribución e integración de sistemas existentes, pero en nuestro caso, no constituye en sí, la solución a los problemas de la gestión de fallos. Sin embargo, es útil realizar un desarrollo distribuido que posteriormente pueda ser integrado en un sistema CORBA.

2.5 Conclusiones

Tomando como base los resultados de estos análisis podemos concluir que un sistema de gestión de alarmas eficiente debe presentar las siguientes características:

- Debe disponer de facilidades de configuración y una GUI sencilla e intuitiva. Además, esta interfaz de usuario debe permitir el acceso al sistema desde diversas localizaciones de la red a gestionar y no depender únicamente de un gestor central. Esta filosofía es la adoptada en muchos sistemas como Minerva, MANDATE o NETMODE. El desarrollo en capas del sistema de gestión y la implementación de APIs de alto nivel que encapsulen las operaciones de gestión subyacentes, simplifica enormemente el desarrollo y ampliación del sistema de gestión.
- Facilidad de expansión y escalabilidad de agentes y de sus prestaciones, sin comprometer el rendimiento global de la red a gestionar. Para esto debemos considerar siempre un sistema con arquitectura distribuida basada en alguna técnica de movilidad de código. El uso de sistemas basados en delegación de *scripts* puede resultar adecuado para construir sistemas dinámicamente expansibles, pero presenta dos inconvenientes fundamentales; en primer lugar, el agente que delega el *script* no tiene control sobre la ejecución del mismo, de forma que si se bloquea no puede suspenderlo. En segundo lugar, las políticas de gestión que pueden establecerse mediante este método residen en el *script* y han de ser definidas previamente. El uso del paradigma de delegación puede solventar en parte estos problemas, entre los que se encuentra el de microgestión entre agente y gestor presente en entornos tradicionales.
- El rendimiento de los sistemas distribuidos es mejor que el de los centralizados, aún en el caso de delegación de código o *scripts*. En un sistema de gestión de alarmas, la información acerca de los eventos siempre es muy superior al necesario intercambio de código entre las entidades involucradas. Los sistemas basados en delegación permiten controlar la carga de procesamiento de los distintos agentes, siendo posible la creación de agentes genéricos de bajo peso computacional. En estos casos, la adopción de modelos *multi-thread* para la incorporación de nueva funcionalidad se revela como la mejor opción, dado que implica una interacción síncrona entre las entidades involucradas.
- Un sistema de gestión debe garantizar la integridad de la información acerca de los eventos. El control de acceso a los GUI, y los mecanismos de autenticación y encriptado de primitivas de comunicaciones permite mantener esta información segura frente a ataques de intru-

sos o de usuarios maliciosos. Es fundamental que el sistema identifique el origen de cualquier petición o notificación y que permita el encriptado de la información que circula en estas primitivas de gestión.

- En una red de telecomunicaciones moderna conviven gran cantidad de protocolos de gestión. El sistema de gestión de alarmas debe permitir la integración de alarmas de distintos protocolos de manera transparente al operador. Además, debe ofrecer al usuario, una visión estandarizada de las mismas para que su recepción y análisis resulte más fácil. Otra de las cualidades que debe tener un sistema de gestión es la capacidad de acceder e integrar aplicaciones previamente desarrolladas. Esta característica es fundamental cuando se quiere modernizar un sistema sin que ello suponga un abandono de muchas de las funcionalidades desarrolladas explícitamente para modelos antiguos. El uso de técnicas de distribución como CORBA facilitan el desarrollo de este tipo de sistemas.
- Es fundamental que el sistema de gestión de alarmas sea fiable y pueda seguir realizando tareas de gestión en casos de fallo, que es cuando realmente se le necesita. En este sentido, la gestión por delegación ofrece la posibilidad de autonomía de los agentes. Una vez que el código ha sido delegado, reside en el agente por lo que puede seguir realizando las labores de gestión de manera independiente del proceso que lo ha delegado. Las alarmas graves, que lleven asociado una acción como respuesta, podrán en la mayoría de los casos ser atendidas a pesar de condiciones anómalas o de no conectividad entre los diversos nodos del sistema.
- La información acerca de los eventos y su posterior registro debe seguir un modelo estandarizado de forma que sea posible el acceso a esta información por diversas plataformas ya existentes en el mercado. Además esta información debe aportar al operador la semántica necesaria para determinar el origen y la localización del fallo. El sistema debe facilitar también el tratamiento eficiente de históricos, por lo que la opción de la adopción de un gestor de base de datos para realizar esta tarea, se convierte en la mejor posibilidad. Estas bases de datos permiten una gran flexibilidad de acceso de forma eficiente y desde diversos lugares de la red. Además entorno al gestor de base de datos pueden implementarse mecanismos y procedimientos que impliquen la colaboración de distintas entidades de gestión.

3 .- *Descripción de la arquitectura propuesta*

3.1 Introducción.

En este capítulo se describe la arquitectura propuesta para la gestión de alarmas siguiendo el paradigma de la gestión por delegación ([Carneiro, 98a]; [Carneiro, 98b]; [Carneiro, 98c];). La arquitectura es analizada desde tres puntos de vista:

- *Arquitectura de soporte*: Constituida por los componentes que permiten la delegación de código entre diversos agentes. Esto va a facilitar la construcción de un sistema totalmente distribuido, con gran facilidad de expansión y escalabilidad.
- *Arquitectura de comunicaciones*: Componentes y primitivas que permiten la comunicación entre los diversos procesos y módulos del sistema.
- *Arquitectura de información*: Soporte de información tanto para las alarmas, como para los procesos que pueden ser delegados.

La arquitectura propuesta toma como referencia la gestión basada en delegación, que a su vez, tiene como concepto central la utilización de agentes extensibles. Un agente extensible o elástico, es un programa que puede ser delegado, extendido y/o contraído dinámicamente mientras está siendo ejecutado bajo control remoto del proceso que lo ha delegado.

Esta arquitectura surgió con la perspectiva de solventar alguna de las deficiencias que presenta la tecnología de agentes móviles, mucho más madura y extendida. Usamos el término agente móvil para describir un programa que es dinámicamente planificado por un proceso remoto, en tiempo de compilación y ejecución. Para conseguir esto, deben habilitarse mecanismos que permitan realizar esta operación de planificación remota, posibilitando el acceso a los recursos locales por parte del agente móvil. También deben proporcionarse mecanismos que permitan controlar esta ejecución. En tercer lugar, es necesario garantizar que los agentes no comprometen los recursos de la máquina local, tanto en seguridad como en rendimiento. En los últimos años, se han propuesto varias tecnologías para el desarrollo de código móvil, entre las que se puede destacar *Telescript* [White, 94] y *Safe-TCL* [Borenstein, 94]. Estas propuestas tienen una base común, los

agentes son programas *script* que son enviados a un intérprete remoto que los ejecuta. En el entorno de JAVA [JAVA, 95] también existen plataformas que facilitan la movilidad de código con características como la creación remota de objetos, el envío remoto de mensajes a esos objetos, control de tiempo de vida del agente, movilidad de objetos y agentes (sin ejecución), migración programada (pudiendo especificar itinerarios), persistencia de objetos y posibilidades de mensajería y eventos distribuidos. Estas plataformas cuidan en detalle la integración con otras aplicaciones, habilitando servicios CORBA [OMG 91.12.1] para esta finalidad.

Las diferencias existentes entre las tecnologías de agentes móviles y gestión por delegación podemos resumirlas del siguiente modo:

1. En los sistemas basados en agentes delegados, los agentes pueden ser desarrollados en cualquier lenguaje compilado o interpretado utilizando los actuales lenguajes de programación y sus potentes herramientas de desarrollo. En la mayoría de los casos, el uso de código móvil viene condicionado por la utilización de un determinado lenguaje de programación.

2. En los sistemas basados en procesos elásticos, puede delegarse un interprete enviarle *scripts* delegados para su ejecución. Por ejemplo, si necesitamos ejecutar un *script* TCL para llevar a cabo una función determinada podemos delegar primero el intérprete y posteriormente el programa interpretado. Una vez que finaliza la ejecución del programa *script*, tanto el intérprete como el *script* pueden ser abandonados. Por tanto, el procesamiento elástico proporciona soporte para los agentes basados en *scripts* como caso particular.

3. Los procesos elásticos pueden extender los mecanismos de control de acceso del sistema operativo y aplicarlos para conseguir agentes seguros. Por ejemplo, un agente delegado puede ser ejecutado únicamente con unos permisos determinados que le impidan acceder o dañar ciertos recursos de la máquina local.

4. El modelo de delegación remota extiende las capacidades de control de la ejecución, lo que facilita la creación de modelos flexibles para controlar el agente. Si un proceso del agente no responde o se ha quedado bloqueado es posible terminarlo remotamente. En un sistema basado en *scritps*, esto no es posible.

5. Distribución de la funcionalidad dinámicamente, dado que se puede distribuir el software de gestión entre los diversos dispositivos, extendiendo la funcionalidad según se requiera. Las aplicaciones de gestión pueden dinámicamente adaptarse y cambiar según la sobrecarga de los recursos en los que se está ejecutando.

6. Autonomía e interacciones de tiempo real: La gestión por delegación puede controlar dinámicamente la granularidad de las interacciones entre las aplicaciones que se están ejecutando en una plataforma y un dispositivo de red. Pueden realizar operaciones sobre los propios datos en los dispositivos, acceder a los recursos de manera abierta. Además, es posible realizar tareas de gestión autónomas mientras no existen conectividad con el centro de gestión de red.

3.2 Arquitectura de soporte

El concepto fundamental en el diseño de la arquitectura de gestión de alarmas es la utilización del concepto de proceso elástico (PE en adelante), ya introducido con anterioridad. Un proceso elástico es un proceso *multithread*, capaz de realizar la carga dinámica de *threads*, soportar el enlace dinámico de estos *threads* dentro de su espacio de direcciones y realizar comunicaciones remotas con instancias de procesos que se encuentran en su proceso elástico local o en otros procesos elásticos remoto. Todo ello dentro de un entorno de control remoto de la ejecución de los *threads*. Estos cambios, internos al proceso elástico, son el resultado de interacciones explícitas con otros procesos elásticos.

Para que el sistema soporte la ejecución de procesos elásticos, es necesario disponer de un protocolo de delegación remota y de una arquitectura para el soporte de delegación. El protocolo de delegación remota permite el intercambio de mensajes y código entre los diferentes procesos elásticos, mientras que la arquitectura para el soporte de delegación es la encargada de procesar y aceptar estos mensajes. Por tanto, un proceso elástico utiliza los servicios de la arquitectura de soporte de delegación para extender/contraer su funcionalidad dinámicamente, mediante la inclusión/eliminación en su espacio de direcciones de una imagen de un código delegado (PDI en adelante) y el protocolo de delegación remota para intercambiar mensajes entre procesos internos y/o externos. El protocolo de delegación remota se expone en el apartado de arquitectura de comunicación del presente capítulo.

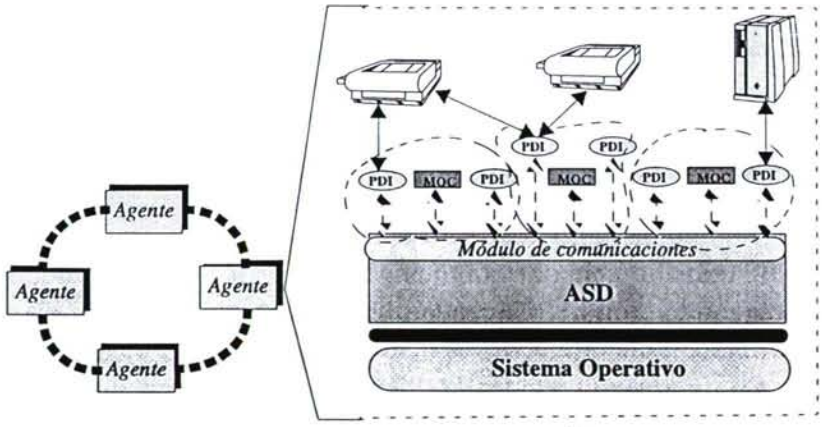

Figura 3.1. *Estructura de un proceso elástico.*

A continuación se describe la arquitectura para el soporte de la delegación (ASD en adelante) que constituye el núcleo de los procesos elásticos. En la figura 3.1 se muestra la arquitectura típica de un proceso elástico. Puede apreciarse que varios procesos elásticos hacen uso del mismo módulo de soporte a la delegación (ASD) con la finalidad de establecer distintos mecanismos simultáneos de gestión sobre uno o varios dispositivo. Cada proceso elástico cuenta con un Módulo de Control (MOC en adelante) que es el encargado de controlar todo el entorno de ejecución del proceso delegado, este módulo que será expuesto posteriormente, contiene el conjunto de ordenes necesario para asegurar la estabilidad del proceso elástico, así p.e. será el encargado de ejecutar los PDIs necesarios para garantizar la autonomía del agente cuando no existe comunicación con el resto de módulos de la arquitectura. Como se aprecia en la figura 3.1 un PDI puede controlar más de un dispositivo simultáneamente con lo que es fácil desarrollar programas para la gestión de agrupaciones de dispositivos del mismo tipo. Otra posibilidad es que un dispositivo sea gestionado simultáneamente por múltiples PDIs. Esta última facilidad permite establecer mecanismos de

seguridad en el acceso a los dispositivos, dependiendo del usuario que haya ordenado la delegación del correspondiente PDI, tendrá permisos para realizar determinadas operaciones o no (estos permisos los determina el módulo de control).

Por tanto, este modelo añade a las tradicionales facilidades que ofrecen los sistemas distribuidos y el código móvil, la facilidad multiagente de los procesos elásticos. Esto permite disponer de múltiples agentes en un recurso, compartiendo las facilidades del soporte de delegación.

Para que todo esto sea posible, se ha desarrollado una arquitectura para el soporte de delegación (ASD) que permite las siguientes funcionalidades:

- *Instanciar PDIs*:

  La arquitectura permite instanciar código que recibe de otros módulos del sistema. Esta instanciación consiste en crear una imagen en ejecución del código que recibe. Utiliza los servicios de delegación remota para permitir al módulo que ha delegado el código mantener el control sobre el mismo, siempre mediante intercambio de mensajes. De este modo, si la instancia se bloquea o no responde, el módulo que ha ordenado la delegación puede terminar su ejecución. Si ocurre una falta de conectividad con ese módulo delegador, el módulo de control (MOC), responsable del proceso elástico puede tomar el control sobre la instancia y ordenar su suspensión o terminación. Como se ha indicado anteriormente, cada una de estas instancias recibe el nombre de PDI.

- *Delegar PDIs*:

  Permite la comunicación con otra ASD, para facilitar el intercambio de código entre las dos. Tiene en cuenta consideraciones de seguridad y control de acceso. Para esta finalidad utiliza los servicios de delegación remota, cuyas primitivas permiten la transferencia de código entre ASDs. El PDI delegado es instanciado remotamente por el MOC responsable del proceso elástico. El ASD es capaz de determinar si ese código ya se encuentra en el destino, evitando transferencias innecesarias de código. Una vez delegado, el código reside remotamente en la ASD destino hasta que se terminan todas sus instancias.

- *Comunicación entre PDIs*:

  Se habilita un mecanismo que permite la comunicación directa entre PDIs (mediante el envío de mensajes asíncronos o comunicación memoria-memoria), tanto entre los que se ejecutan bajo el mismo ASD, como entre PDIs de distintos ASDs. Se establece un único mecanismo de transporte válido para las dos alternativas. Cada ASD conoce todos los PDIs locales (mediante una tabla denominada *PD-Table*) y los ASD con los que tiene conexión (mediante la *ASD_Table*). La *PD_Table* es mantenida por el ASD local, mientras que la *ASD_Table* la mantiene un módulo bien conocido de la arquitectura, el gestor de configuración (MCF). Para conocer la ubicación de un determinado PDI, se utiliza el identificador del mismo, que consta de varios campos que permiten identificar unívocamente en donde se está ejecutando una instancia de código.

- *Solicitud de delegación*:

  Cualquier ASD dispone de la posibilidad de solicitar la delegación de PDs de otros ASDs vecinos. Esto se realiza mediante una primitiva que tiene como respuesta, si la operación es posible, una delegación por parte del ASD remoto. Esta solicitud de delegación es útil para permitir a los PDIs que puedan expandir su propia funcionalidad con nuevos procesos delegados. El módulo de control utiliza esta capacidad, para solicitar a ASDs bien conocidos, los procesos delegados que necesita para operar sobre el dispositivo a gestionar.

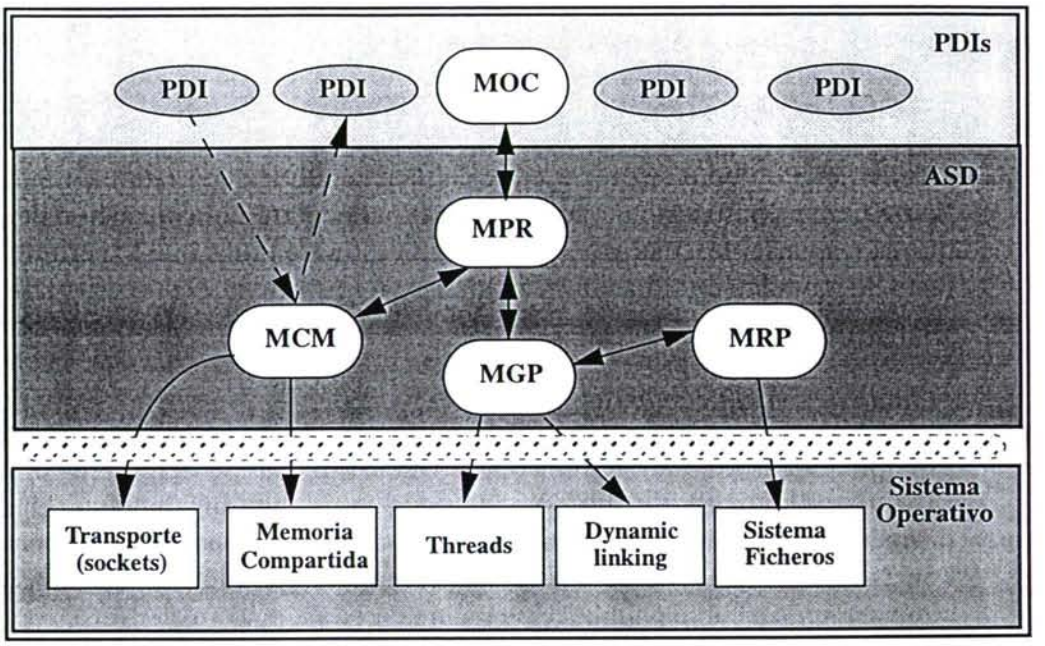

Figura 3.2. *Estructura interna de un ASD.*

La estructura interna completa de un ASD puede apreciarse en la figura 3.2. Consta de tres niveles que se describen a continuación:

- *Capa de aplicaciones*: En donde se ejecutan las instancias de los procesos que han sido delegados. Estos procesos se intercambian mensajes a través de un servicio de transporte ofrecido por el módulo de comunicaciones (MCM) y su ejecución es supervisada por el módulo de control (MOC) responsable de su ejecución.

- *Capa ASD*: Proporciona los servicios básicos de delegación. Facilita la integración de PDIs en el proceso elástico, posibilitando la instanciación y control de la ejecución remota de los PDIs, que pueden ser almacenados localmente de manera provisional. Constituye el núcleo del proceso elástico y se explica posteriormente con más detalle.

- *Sistema Operativo*: Proporciona los servicios del sistema operativo que faciliten el acceso a los recursos gestionados. Este acceso se hace mediante una librería de rutinas que separa físicamente el sistema operativo subyacente con la capa ASD. De este modo se independiza el ASD de la plataforma en la que se ejecuta, posibilitando las facilidades de código móvil, cuyo soporte debe instalarse a este nivel. Esta librería se implementa como un demonio (*ASD_daemon*) que contiene la lógica necesaria para recibir y ejecutar el resto del ASD en la máquina en la que reside.

Dentro de la capa ASD pueden apreciarse los siguientes módulos:

- *Módulo principal (MPR)*:

  Inicializa el entorno de ejecución del *ASD*, y constituye el núcleo del ASD. Es el encargado de dar respuesta a las primitivas básicas que implementa el Servicio de Delegación Remota (SDR). Mantiene las tablas que le permiten identificar a todos los PDIs del sistema (incluidos los ASDs), controla la ejecución de los PDIs y consta de un gestor de fallos que avisa al módulo de comunicaciones de anomalías que puedan ocurrir durante la instanciación y ejecución de PDIs. También es el encargado de garantizar los servicios de control de acceso sobre los PDIs, con la información que tiene de cada proceso.

- *Módulo de Control (MOC)*:

  Contiene el conjunto de órdenes que definen la política de gestión del proceso elástico. Envía primitivas al módulo principal para controlar la ejecución de los PDIs que es responsable. A su vez mantiene un gestor de fallos que le permite avisar al resto de módulos de situaciones anómalas en la gestión del proceso elástico.

- *Módulo de comunicaciones (MCM)*:

  Permite tanto la comunicación entre PDIs del mismo proceso elástico, como entre PDIs dependientes de distintos módulos de control. Permite también el intercambio de código y primitivas entre ASDs. En casos de fallo, reporta su salida al MOC y envía una alarma de fallo para indicar la anomalía. Es capaz de establecer una comunicación estable para la delegación de PDIs a/por otros ASDs. Contiene una copia de la *ASD-Table*. Esta tabla es gestionada por un proceso externo (Módulo de Configuración) que envía mensajes de actualización a todos los MOC, cuando ocurre alguna anomalía, el MOC envía una alarma de notificación del fallo.

- *Módulo gestor de PDIs (MGP)*:

  Permite la instanciación, planificación y control de los distintos PDIs. Debe proteger p.e. el acceso concurrente de dos PDIs al mismo conjunto de datos (región crítica). Permite las operaciones de instanciación, suspensión de la ejecución, reinicio de la ejecución, terminación y borrado de la copia del PDI.

- *Módulo repositorio (MRP)*:

  Almacena instancias de PDIs que han sido terminadas pero no eliminadas. También almacena aquellas rutinas que pueden ser ejecutadas localmente bajo determinadas causas de fallo e inicialización que le son delegadas por el módulo de control (MOC). Habilita el mecanismo de transporte necesario para enviar/recibir los códigos delegados a través del Módulo de Comunicaciones (MCM).

A continuación se describe más en detalle, cada uno de los módulos que componen el ASD.

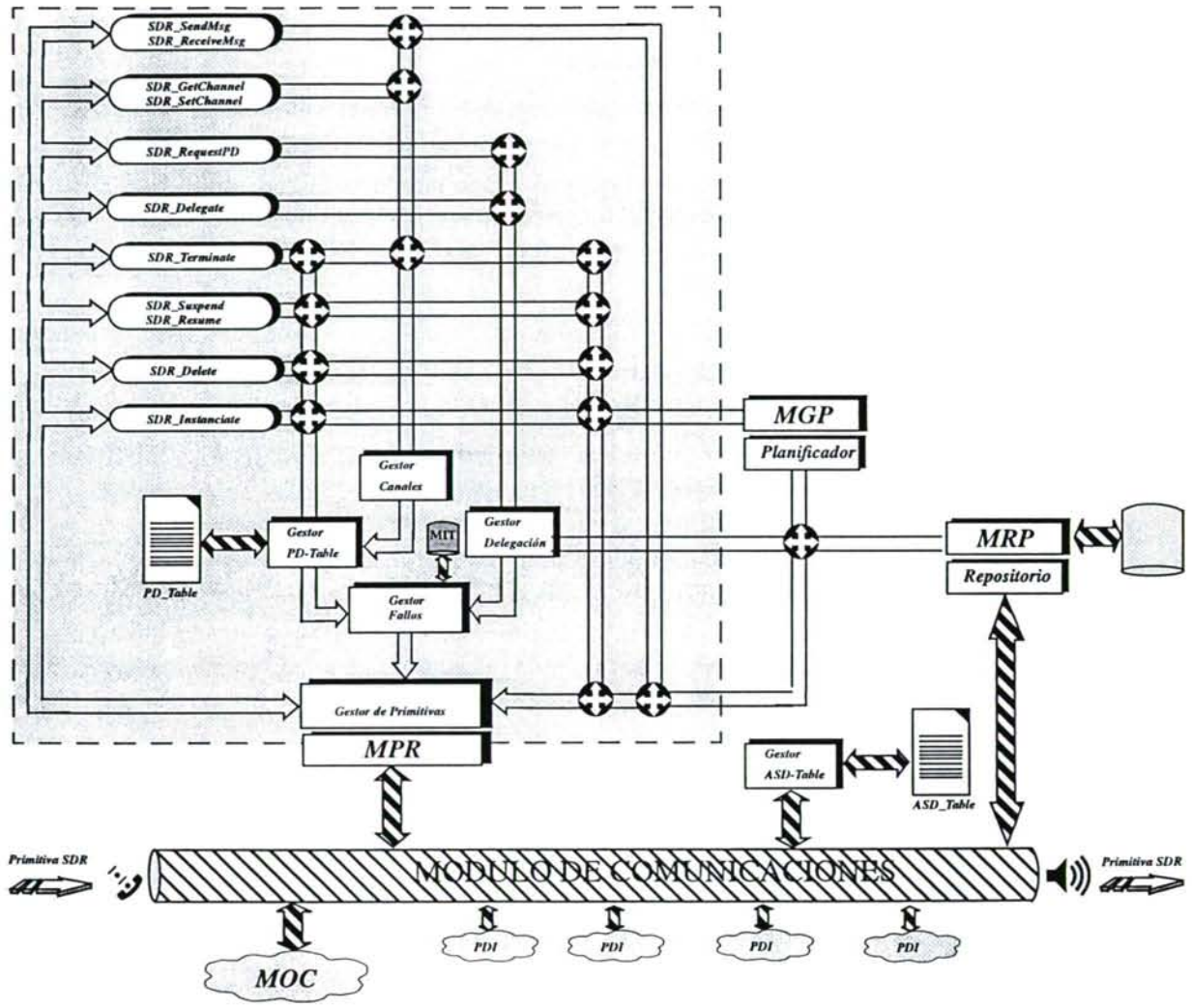

Figura 3.3. *Estructura del módulo principal.*

3.2.1 Módulo principal (MPR).

Este módulo es el encargado de inicializar el ASD y posteriormente dar respuesta a las primitivas que implementa el servicio de delegación remota. También es el encargado de gestionar los canales de comunicación entre procesos y mantener actualizadas la tabla de procesos (*PD_Table*) y la tabla de ASDs (*ASD_Table*). Su estructura es la mostrada en la figura 3.3 y en ella pueden apreciarse los siguientes módulos:

- *Gestor de Primitivas*: Recibe las primitivas del módulo de comunicaciones (MCM) y las envía al correspondiente intérprete de primitivas. Gestiona una cola de primitivas para controlar los flujos de entrada y salida de las mismas. Es el encargado de enviar las primitivas al resto de PDI del sistema, utilizando para ello, al módulo de comunicaciones (MCM).

- *Gestor de PD-Table*: Gestiona una tabla de procesos delegados. Esta tabla se utiliza para conocer los datos más relevantes de todos los PDIs que están siendo ejecutados en el ASD local. Cada entrada de la tabla consta de los siguientes campos.

- *Identificador*: Identificador unívoco del proceso delegado (PDId en adelante). Está compuesto por subcampos con la siguiente organización:
  - ASD en el que reside el proceso que ha ordenado la delegación del PDI. Se utilizan 2 bytes para esta tarea.
  - MOC responsable de su ejecución. Se utilizan 3 bytes, los dos primeros indican el ASD en el que se encuentra el MOC responsable y el último byte indica el identificador de proceso delegado que se corresponde con ese MOC. El módulo responsable de un MOC es el módulo principal (MPR). En este caso se coloca un cero en el último bye. Otra particularidad es que, el primer MOC que se instancia en un ASD siempre recibe el número 1.
  - Código de proceso delegado. Es un identificador de 4 bytes, los dos primeros indican el ASD en el que el código está almacenado (ASDid en adelante) y los dos últimos el identificador global del código (CDid en adelante).
  - PDI: Ocupa 1 byte, e indica el número de instancia del código delegado que se corresponde con este PDI.
- *Propietario*: Indica el usuario propietario del código delegado.
- *Grupo*: Grupo propietario del código delegado.
- *Permisos:* Se trata de nuevo de un campo compuesto de tres subcampos, uno para permisos del propietario, un segundo para permisos de grupo y por último otro para permisos de cualquier del sistema. Cada campo ocupa dos bytes y contiene la activación o desactivación de permisos de Instanciación, Suspensión, Reinicialización, Terminación, Borrado, Delegación, Solicitud de Delegación y Modificación de canales. Así, p.e. 10100010 (0xa2) indica para un subcampo determinado que los usuarios tienen permiso de instanciación del proceso delegado, reinicialización si está suspendido y que puede solicitar la delegación de ese proceso. Cualquier otra operación sobre el PDI no puede ser realizada.
- *Estado*: Estado en que se encuentra el proceso delegado: (I)nstanciado, (S)uspendido o (T)erminado.
- *Canal de entrada*: Descriptor del canal por el que el módulo está escuchando mensajes de otros PDIs.
- *Canal de salida*: Descriptor del canal por el que el módulo está enviando mensajes a otros PDIs.

- *Gestor Canales*: Consulta en la *PD_Table*, la utilización de canales para asignar o informar a un PDI del uso que puede hacer de un canal. Si ocurren errores se emite una alarma que debe ser interpretada por el módulo de comunicaciones (MOC) implicado.

- *Gestor de Delegación*: Encargado de atender a los servicios de delegación y solicitud de delegación. Indica al repositorio la necesidad de recuperación de una instancia que previamente había sido terminada o el almacenamiento de una nueva.

- *Gestor Fallos*: Este módulo es el encargado de verificar si se ha producido un error por sobrecarga de recursos, envío incorrecto de mensajes, etc. Envía una alarma al módulo de control responsable del PDI que ha originado el fallo. Contiene una MIT de alarmas en formato X.733, como citaremos más adelante. Puede solicitar la delegación de código para llevar a cabo la acción asociada a la alarma que se ha generado con el fallo.

3.2.2 Módulo de control (MOC).

El módulo de control es el responsable de iniciar y controlar el conjunto de PDIs que forman el proceso elástico. Acepta las órdenes de que le llegan de los procesos delegadores y se la remite al módulo principal (MPR) para su ejecución. En caso de fallo emite una alarma por su canal de salida, normalmente conectado a algún visualizador de alarmas. Es el responsable de detectar posibles estados de no-conectividad, durante los cuales redireccionará las salidas exteriores a un fichero de log para su posterior reenvío. También, durante estos periodos, puede llevar a cabo operaciones autónomas que permitan controlar el recurso bajo la ausencia de órdenes de algún gestor. La figura 3.4 muestra la estructura interna de este módulo, en el que pueden apreciarse los siguientes componentes:

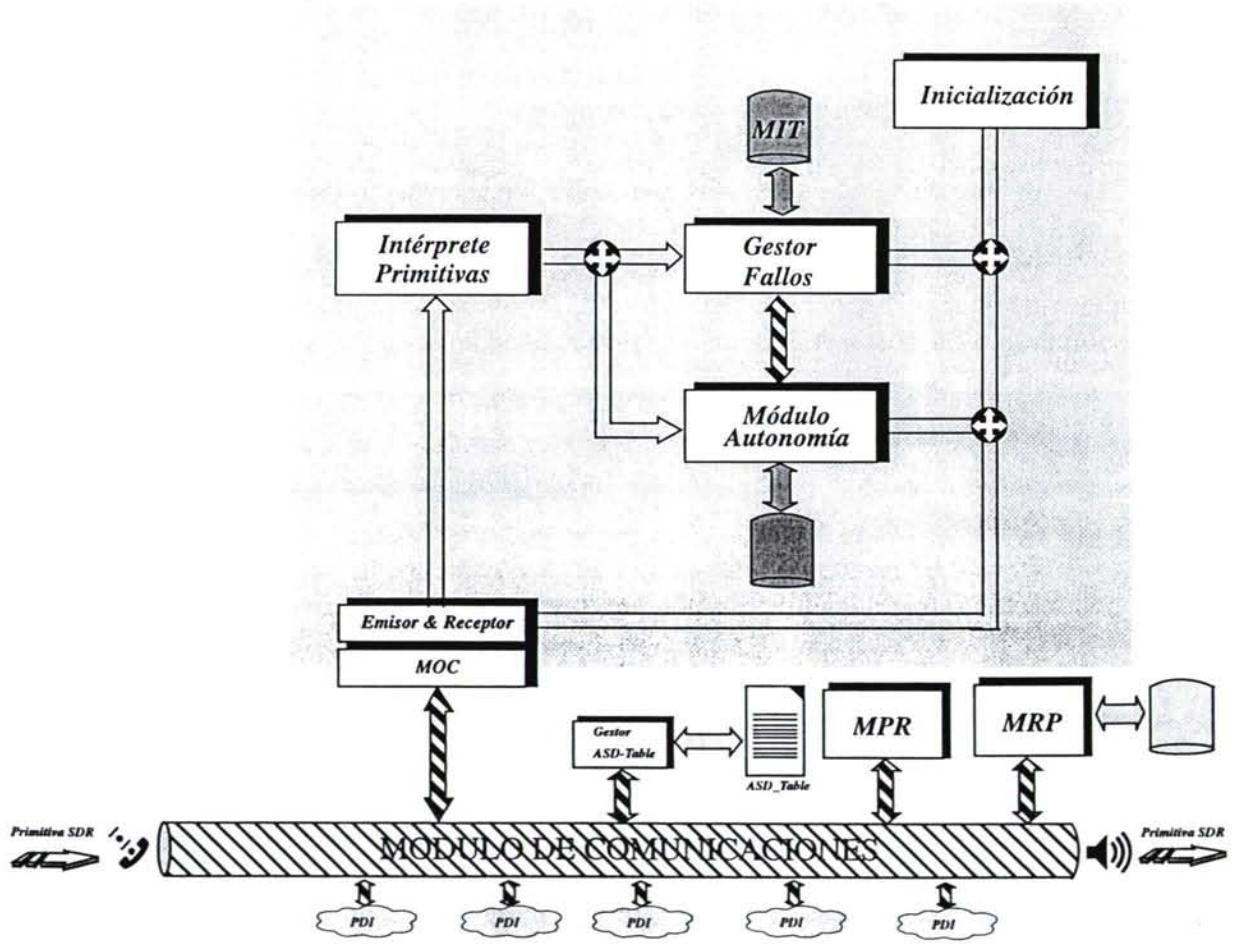

Figura 3.4. *Módulo de control (MOC)*

- *Inicialización*: Durante esta fase, el módulo de control envía primitivas de solicitud de código a los ASD conocidos (para ello consulta la *ASD_Table*). Hace uso del servicio de delegación remota para invocar estas órdenes. Este módulo es el responsable de instanciar los PDI mínimos que requiere el dispositivo a gestionar.

- *Gestor de fallos*: Este módulo se encarga de resolver los fallos que puedan ocurrir durante la gestión del dispositivo. Pueden deberse a violación de privilegios, recursos, instanciación de PDIs, no conectividad del módulo de comunicaciones, etc. Cada evento lleva asociada una alarma que se almacena en una *Management Information Tree* (MIT).

- *Gestor de primitivas*: Encargado de dar respuesta a los mensajes que llegan como órdenes de control sobre algún PDI que se está ejecutando. Comprueba, mediante la copia que tiene de la *PD_Table*, que esa operación puede realizarse y se la envía al módulo principal para su ejecución.

- *Módulo de autonomía*: Este módulo contiene el código que va a ser instanciado durante estados de fallo o no conectividad con el resto de módulos del sistema. El módulo de comunicaciones es el encargado de avisar, a través de una alarma, a todos los módulos de control de la ocurrencia de un aislamiento, como acción asociada a este tipo de alarma, el módulo de control instancia el código que tiene almacenado en un repositorio interno y que realiza las operaciones de gestión durante ese periodo de fallo.

3.2.3 Módulo de comunicaciones (MCM).

Este módulo es el encargado de realizar las comunicaciones entre los distintos PDI que forman parte del agente. La comunicación puede ser interna o externa, y para esto último, utiliza una copia del *ASD_Table* en la que puede obtener la localización del resto de ASDs del sistema. Esta tabla contiene información acerca de todos los ASDs de los que el módulo de comunicaciones tiene conocimiento. La información por cada entrada de la tabla es la siguiente:

- *Identificador*: Número de dos bytes que identifica unívocamente en todo el sistema al ASD (ASDid en adelante).
- *Nombre*: Descripción textual del ASD para indicar el tipo y la funcionalidad del correspondiente ASD.
- *Dirección de protocolo*: Dirección de la máquina en la que se encuentre ejecutando el ASD. Depende del protocolo de red elegido, generalmente IP.
- *Canal de escucha*: Dirección del canal por el que el ASD indicado puede recibir órdenes o código.

Este módulo tiene acceso a las funcionalidades internas de los PDIs que se ejecutan en el entorno de un ASD determinado. Controla los canales de los PDIs, el envío de mensajes, la espera de eventos o cualquier otra característica contemplada en la lógica soportada por los PDIs. Utiliza los servicios del sistema operativo, como *sockets* o memoria compartida, para el intercambio de mensajes entre PDIs, y lo hace de una manera uniforme, tanto si se trata de PDIs locales como entre PDIs que se ejecutan en PDIs diferentes. Este módulo actúa como un *proxy* para todas las órdenes que se intercambian entre PDIs.

3.2.4 Módulo gestor de PDIs. (MGP).

Este módulo permite a procesos remotos instanciar, terminar, suspender o reinicializar PDIs. Cuando un proceso (local o remoto) quiere instanciar un proceso delegado, debe emitir una orden de instanciación al módulo principal que a su vez se la envía al módulo gestor de PDIs. Este módulo consta de un planificador, encargado de planificar la ejecución de los PDIs, determinando

las correctas transiciones entre sus estados de ejecución. Protege el acceso concurrente a regiones críticas, al prevenir que los *threads* (PDIs) pueden acceder concurrentemente a estructuras de datos compartidas. Las zonas de memoria que utilizan los distintos PDIs son bloqueadas por el módulo gestor de PDIs para impedir estos problemas de integridad.

3.2.5 Módulo Repositorio (MRP).

Se encarga de almacenar el código de los procesos delegados que han sido instanciados por algún PDI. Esto facilita que el código no tenga que circular por la red, cada vez que es delegado. Si existe una copia en el repositorio remoto, no se traslada por la red. Proporciona una base de datos común para almacenar datos en el sistema de fichero de bajo nivel. Dispone de una interfaz que permite almacenar, bloquear y borrar PDIs. Por ejemplo, si el módulo de comunicación recibe una orden de delegación, almacena un fichero intermedio en el repositorio. Si la delegación es completada, el repositorio devuelve un identificador de la instancia y se registra la misma en la *PD_Table*. El repositorio proporciona un servicio de nombrado que permite identificar a los PDIs con el nombre del fichero en el sistema de ficheros. Contiene una estructura de datos por cada PDI, en el que se almacena, además del código, un contador de uso e información de autenticación y seguridad acerca del proceso delegado.

3.3 Arquitectura de comunicaciones

La arquitectura de comunicaciones está formada por un conjunto de primitivas que constituyen el servicio de delegación remota. Mediante este servicio, los agentes pueden intercambiar mensajes y controlar su ejecución. Para esta tarea utiliza los servicios del módulo de comunicaciones que facilita el intercambio de primitivas entre los diversos PDIs del sistema. Las primitivas que pueden realizarse se agrupan de la siguiente forma:

3.3.1 Primitivas de control de ejecución de PDIs.

Mediante estas primitivas los procesos que han delegado código pueden controlar su ejecución remota. Las primitivas que pueden utilizarse son las indicadas en la siguiente tabla:

Primitiva	Descripción
SDR_Instanciate(PDid_org CDid, PDid_dest)	Permite instanciar un código de proceso delegado identificado por **CDid**, bajo la supervisión de un determinado módulo de control, indicado en **PDid_dest**. Como resultado de esta instanciación se añade una nueva entrada en la *PD_Table* y se le asigna un nuevo PDid a la instancia. Si ocurre un error se genera una alarma por el gestor de fallos. El ASD destino comprueba que el **PDid_org** tiene permiso para llevar a cabo esta operación.

Primitiva	Descripción
SDR_Terminate(PDid_org PDid)	Permite finalizar la ejecución de una instancia de un determinado proceso identificado por **PDid**. El proceso delegado puede volver a ser reinicializado por completo sin necesidad de una nueva delegación. Se actualiza la *PD-Table*, modificando el estado del proceso. El ASD destino comprueba que el **PDid_org** tiene permiso para llevar a cabo esta operación.
SDR_Suspend(PDid_org PDid)	Suspende temporalmente la ejecución de un determinado proceso identificado por **PDid**. Este proceso puede ser reinicializado con la primitiva SDR_Resume. Durante el tiempo que se encuentra en este estado se suspende la planificación del *PDi*. Actualiza la *PD-Table*, modificando el campo estado. El ASD destino comprueba que el **PDid_org** tiene permiso para llevar a cabo esta operación.
SDR_Resume(PDid_org PDid)	Activa de nuevo la ejecución de un PDI, que había sido suspendido con el comando SDR_Suspend. Devuelve el *PDi* al estado activo y planificable. Actualiza la *PD-Table*. El ASD destino comprueba que el **PDid_org** tiene permiso para llevar a cabo esta operación.
SDR_Delete(PDid_org CDid)	Elimina las instancias de un PD y elimina del repositorio el código almacenado, que está identificado por **CDid**. Para instanciar otra vez ese código en el ASD es necesario una nueva delegación. Todas las instancias del PD finalizan su ejecución. Actualiza la *PD-Table*. El ASD destino comprueba que el **PDid_org** tiene permiso para llevar a cabo esta operación.

3.3.2 Primitivas de Delegación de PDIs.

Primitiva	Descripción
SDR_Delegate(PDid_org CDid, PDid_dest)	Permite el transporte de un código de proceso delegado identificado por **CDid**, a otro ASD, en el que se instanciará bajo la supervisión de un determinado módulo de control, indicado en **PDid_dest**. Se asocia a una orden de instanciación y se almacena el código en el repositorio. Si ocurre un error se genera una alarma por el gestor de fallos. El ASD destino comprueba que el **PDid_org** tiene permiso para llevar a cabo esta operación.

Primitiva	Descripción
SDR_RemoteDelegate(PDid_org CDid, PDid_dest)	Solicita a un ASD intermedio, identificado por **PDid_ASD**, que delegue un código (**CDid**) en otro ASD, bajo la supervisión del módulo de control especificado en **PDid_dest**. Deben chequearse los permisos para la realización de esta operación comprobando que el **PDid_org** puede realizar esta operación.
SDR_Request(PDid_org CDid,)	Solicita al proceso delegado **PDid_dest** que le delegue un código (**CDid**). El proceso destino será **PDid_org** y el módulo de control responsable, el mismo que el de **PDid_org**. Se comprueba que no se violan los permisos necesarios.

3.3.3 Primitivas de Comunicación.

Primitiva	Descripción
SDR_SendMsg(PDid_dest, tipo, Nº fragmentos, fragmento, Msg)	Permite enviar un mensaje (**Msg**) a un determinado proceso delegado (**PDid_dest**). Consulta la *PD-Table* para ver el canal por el que está recibiendo mensajes el PDid destino. Es responsabilidad del destino averiguar el formato del mensaje en base a la variable tipo. En principio existen los tipos: (0) mensaje genérico y (1) alarma en formato X.733. El campo fragmento indica el número de paquete dentro del Nº de fragmentos que componen el mensaje.
SDR_ReceiveMsg(&Msg)	Permite a un PDI recibir mensajes de otros PDIs.
SDR_GetChannel(PDid_org PDid_dest)	Mediante este comando un PDi origen (**PDid_org**) solicita el canal de entrada de otro PDi (**PDid_dest**). Para llevar a cabo esta operación se consulta la *PD_Table* y se estudian los permisos que sobre ese proceso tiene el **PDid_org**.
SDR_SetChannel(PDid_org PDid_dest Channel)	Permite fijar el canal de salida (**Channel**) de un proceso delegado identificado por **PDid_dest**. Se comprueba que el **PDid_org** puede realizar esta operación. Esta primitiva desencadena una actualización de la *PD_Table*.

3.3.4 Primitivas de actualización de tablas.

Primitiva	Descripción
SDR_UpdateASDTable(PDid_org *InfoASD*)	Actualiza la *ASD_Table* con la información completa acerca de una de sus entradas (**infoASD**).Comprueba que el **PDid_org** tiene permiso para modificar los datos de la *ASD_Table*.
tabla **SDR_GetASDTable(** PDid_org)	Obtiene la *ASD_Table* del receptor de la primitiva. Esta primitiva es útil, por ejemplo, cuando se inicializa un ASD o cuando se quiere comprobar la corrección de su tabla.
SDR_SetRecordPDi(*PDid_org,* *datosPDI*)	Añade un registro a la tabla global de PDI del sistema. La entrada a añadir se indica en *datosPDI*. Si ocurre algún error, se envía una alarma.
SDR_RemoveRecordPDI(*PDid_org* PDid)	Elimina una entrada de la tabla global de PDIs. Se comprueba que el proceso delegado origen puede realizar esta operación y si este PDid existe. Si ocurre algún error, se envía una alarma.

3.4 Arquitectura de información.

Cada uno de los módulos que componen el sistema cuentan con un repositorio de información integrada por estructuras tipo X.721 [ITU-T X721] para el almacenamiento de alarmas y estructuras de tipo proceso delegado, para el almacenamiento de código delegable que pertenece a ese módulo.

La figura 3.5 presenta un ejemplo típico de procesos delegados almacenados en uno de los repositorios del sistema. El proceso delegado (GTRAP) ha sido incluido en el sistema, concretamente en el gestor de adaptadores, con los permisos 919101 y pertenece al grupo admin y al usuario victor. Esto indica que tanto *victor* como todos los usuarios pertenecientes al grupo *admin* pueden instanciar, terminar y modificar los canales de salida de cualquier instancia del proceso delegado GTRAP. El resto de operaciones, es decir, suspensión de la ejecución de la instancia, reinicialización de una instancia suspendida, borrado u eliminación de todas las instancias del proceso delegado y solicitud de delegación del proceso de manera remota únicamente están permitidos al

usuario especial *root*. El resto de usuarios del sistema, únicamente puede modificar los canales de salida de cualquier instancia del proceso delegado GTRAP.

Repositorio del gestor de adaptadores

0201	javier	opera	610101	Adapta_P
0202	pedro	opera	e56161	Adapta_I
0203	javier	admin	ffffff	Adapta_G
0204	victor	admin	919101	GTRAP
0205	javier	admin	616161	Adapta_M
0206	root	admin	ffffff	SNMP
0207	root	admin	ffffff	CMIP
0208	victor	admin	919100	Adapta_R

Figura 3.5. *Ejemplo de directorio de procesos delegados*

Las alarmas son almacenadas siguiendo el formato X.733 [ISO 10164-4] para el registro de notificaciones. Para cada alarma se incluyen los siguientes campos:

- Identificador de Invocación (*logRecordId*)
- Clase de Objeto Gestionado (*managedObjectClass*)
- Ejemplar de Objeto Gestionado (*managedObjectInstance*)
- Tipo de Evento (*eventType*)
- Tiempo del Evento (*eventTime*)
- Causa Probable (*probableCause*)
- Problemas Específicos (*specificProblems*)
- Gravedad Percibida (*perceivedSeverity*)
- Situación de Respaldo (*backUpStatus*)
- Objeto de Respaldo (*backUpObject*)
- Indicación de Tendencia (*trendIndication*)
- Información de Umbral (*thresholdInfo*)
- Identificador de Notificación (*notificationIdentifier*)
- Notificaciones Correlacionadas (*correlatedNotifications*)
- Definición de Cambio de Estado (*statechangeDefinition*)
- Atributos Monitorizados (*monitoredAttributes*)
- Acciones de Reparación Propuestas (*proposedRepairActions*)
- Texto Adicional (*additionalText*)
- Información Adicional (*additionalInformation*)

3.5 Modelado de la arquitectura.

En este apartado se introduce una arquitectura para la gestión de alarmas que solventa las dos principales deficiencias de los actuales sistemas de gestión: la centralización del procesamiento y la ausencia de riqueza de información. La arquitectura presenta una estructura totalmente distribuida que permite al operador disponer de un entorno flexible, modular, fácil de configurar y expandir. En la arquitectura pueden apreciarse dos tipos de módulos:

- *ASD específicos*: Estos ASD tienen una dirección bien conocida por el resto ASDs del sistema. Contienen información de interés general para el sistema. Así existe un módulo encargado de la configuración del sistema (gestor de configuración), un módulo para rutinas de filtrado y de *polling* (Gestor de interrogadores), un módulo para la gestión de adaptadores de protocolos (Gestor de adaptadores) y un módulo de visualización de alarmas (Visor de alarmas). Estos módulos se implementan como un ASD al que se le instancia un módulo de control que es capaz de realizar la labor específica del nodo. Esta organización permite que la funcionalidad de gestión se encuentre estáticamente centralizada (cuando no esta siendo ejecutada) y dinámicamente distribuida (en el momento de la ejecución de los procesos delegados). Esta organización facilita también el desarrollo de procesos de gestión globales o de interrelación entre diversos nodos y la construcción de herramientas de faciliten la construcción de procesos delegados especializados, p.e. herramientas gráficas de construcción de filtros.
- *ASD generales*: Son los ASD que se ejecutan en los nodos que dan acceso a los dispositivos a gestionar. En cada uno de estos ASD pueden ejecutarse diferentes módulos de control. Es uno de los ASDs específicos (el gestor de configuración), el encargado de inicializar estos ASDs y transferirles el módulo de control inicial, según un fichero de configuración inicial que es mantenido por los distintos visores de alarmas. Previamente, en cada nodo debe existir un *ASD_daemon* capaz de recibir la delegación del ASD correspondiente.

La figura 3.6 muestra la arquitectura del sistema de gestión en la que pueden apreciarse los siguientes módulos:

- *Visor de alarmas* (MVA): Contiene las rutinas de visualización y almacenamiento de alarmas en formato X.733. Desde este módulo se controla a los dispositivos a gestionar y se emiten órdenes de solicitud de delegación para modificar la funcionalidad de los agentes encargados de esta gestión. Pueden existir en el sistema más de un MVA de diferentes usuarios, Es el encargado de actualizar la tabla global de PDIs y ASDs, mediante acceso al módulo de configuración. Es un módulo especial, dado que puede ser ejecutado desde cualquier lugar y accede directamente al módulo de configuración para determinados servicios, como la actualización de tablas.

- *Gestor de Interrogadores* (MIR): Contiene las rutinas encargadas de interrogar periódicamente a los equipos cuando estos no emiten alarmas. Permite la definición de grafos de estados para la construcción de alarmas. En este módulo también se almacenan los diferentes filtros que pueden ser ejecutados sobre los equipos a gestionar. Con este módulo se podrán definir políticas de *polling* y filtrado sobre grupos de equipos.

- *Gestor de Adaptadores* (MAR): Rutinas de adaptación de protocolos estándares o propietarios a formato X.733. Constituye los puentes que permiten integrar cualquier tipo de alarmas en el sistema (ver un ejemplo en [Carneiro, 97a]). Contiene también los módulos que permiten emitir alarmas en diversos formatos a uno o varios destinos.

- *Gestor de Configuración* (MCF): Contiene los distintos módulos de control (MOC) que pueden ser ejecutados sobre los diferentes equipos. Cada uno de estos módulos actuará sobre el equipo con unos determinados parámetros de seguridad, acceso y prioridad (especificados en su campo de permisos). Es el encargado de gestionar y actualizar todas las *ASD_Tables* de que tiene conocimiento. Además es el encargado de inicializar todo el

sistema.

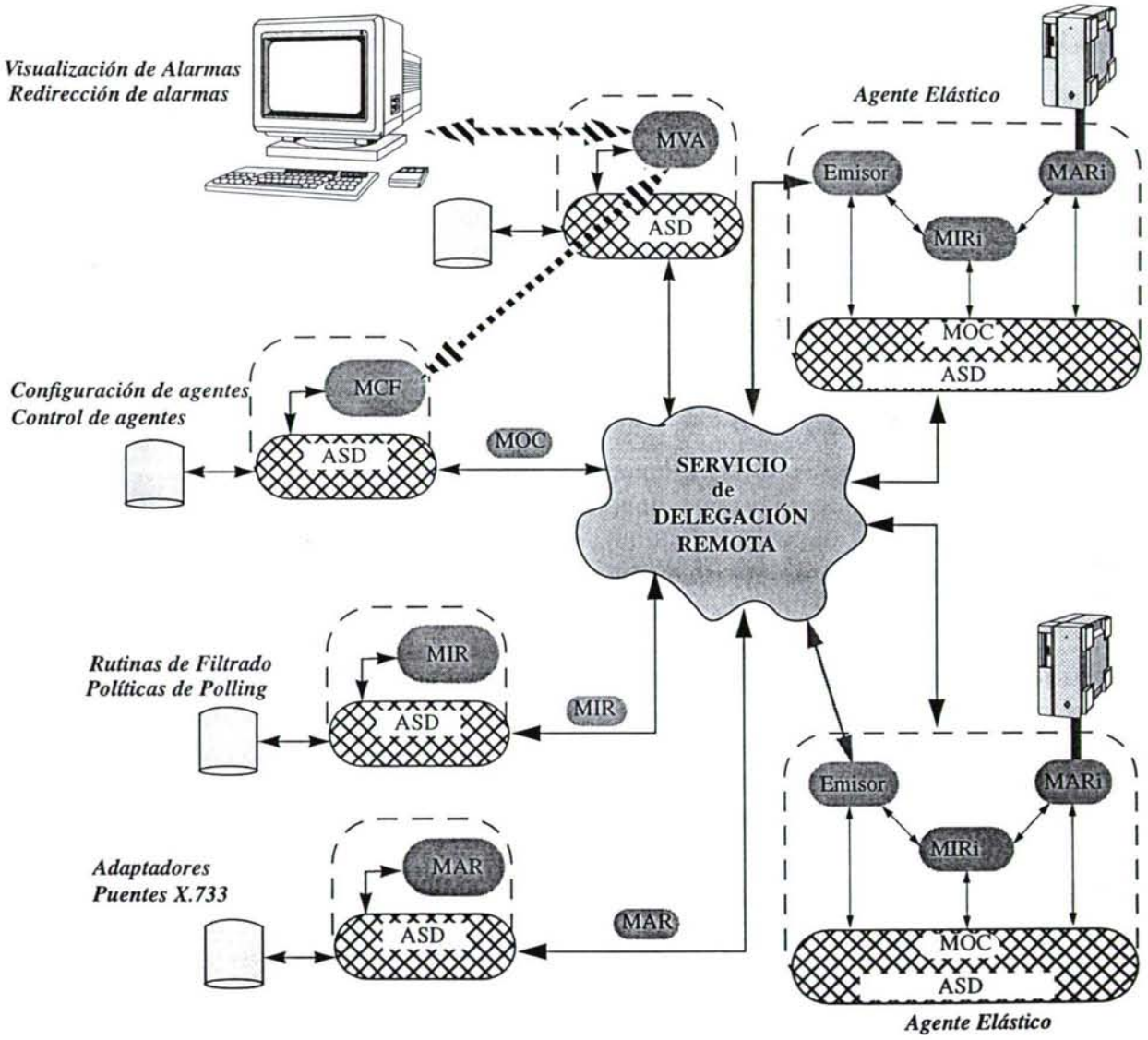

Figura 3.6. *Estructura general del sistema de gestión de alarmas.*

A continuación se desarrolla con mayor detalle cada uno de estos módulos específicos que componen este sistema.

3.5.1 Gestor de Configuración (MCF).

Constituye el núcleo central del sistema de gestión. Es el encargado de inicializar el sistema e instanciar los módulos específicos. Para esta tarea utiliza la información almacenada en la tabla *ASD_Table* inicial. La figura 3.7 muestra un ejemplo de esta tabla. En ella se aprecia que, p.e. el

ASD identificado con el número 02 se corresponde con el módulo de adaptadores y debe ser ejecutado en la máquina 193.144.50.190. Los canales de escucha, señalados con un cero en la tabla, serán asignados en el momento de la inicialización del correspondiente módulo.

01	MCF	193.144.50.190	0
02	MAR	193.144.50.190	0
03	MIR	193.144.50.180	0
04	ASDC1	193.144.50.134	0
05	ASDC2	193.144.50.150	0
06	ASDC3	193.144.50.176	0

Figura 3.7. *ASD_Table inicial*

También es el responsable de instanciar aquellos procesos delegados necesarios para implementar las políticas estáticas predefinidas por el operador global del sistema, es decir, aquellos procesos necesarios para mantener una gestión mínima de la red, aún cuando ningún operador está visualizando alarmas. Para realizar esta tarea, mantiene la información acerca de los ASD que debe inicializar en la *ASD_Table* inicial y los procesos delegados a instanciar en la *PD_Table* inicial. La figura 3.8 muestra un ejemplo de esta tabla. En ella puede apreciarse que es necesario instanciar el proceso con el identificador 010500104. Este identificador contiene la información necesaria para llevar a cabo la instanciación: El ASD que solicita la delegación es el 01 (MCF), su módulo de control responsable, dentro del ASD destino (05) es el 0. Esto quiere decir que se trata de un módulo de control, dado que el 0 indica que su responsable es el módulo principal del propio ASD destino. El código correspondiente a ese módulo de control reside en el ASD 01 y tiene el identificador de código 04. Como resultado de la instanciación de ese módulo de control se actualizan los descriptores de los canales de salida y entrada del módulo instanciado.

010500106	root	admin	ff9101	T	0	0
010510407	root	admin	ff9101	T	0	0
010510408	root	admin	ff9101	T	0	0
010600106	root	admin	ff9101	T	0	0
010610407	root	admin	ff9101	T	0	0
010610408	root	admin	ff9101	T	0	0

Figura 3.8. *PD_Table inicial*

El módulo de configuración es el encargado de mantener las tablas de usuarios y grupos que pueden operar sobre el sistema. Estas tablas son las mostradas en la figura 3.9 y contienen toda la información necesaria para identificar a un usuario en el sistema. Por cada usuario en el sistema existe una entrada en la tabla de usuarios en la que se indica su identificar, su clave de acceso, el grupo principal al que pertenece, su descripción textual y el fichero de configuración en la que se almacena la información acerca de la configuración personalizada de su entorno de gestión. Este fichero de configuración es una copia de la *PD_Table* correspondiente a los procesos que han sido

delegados por el usuario, y le permite retomar su entorno de gestión cada vez que entra de nuevo en el sistema.

Tabla de usuarios

```
root:xxxxxxx:admin:Administrador:root.cfg
victor:xxxxxxx:admin:Victor:victor.cfg
javier:xxxxxxx:opera:Javier:javier.cfg
pedro:xxxxxx:opera:Pedro:pedro.cfg
+
```

Tabla de grupos

```
admin: root, victor, javier
opera: javier, pedro
+
```

Figura 3.9. *Tabla de usuarios y de grupos.*

Este módulo contiene los distintos tipos de módulos de control que pueden ser instanciados en cualquier ASD del sistema. Estos módulos serán instanciados por orden expresa de algún módulo de visualización. Mantiene las tablas anteriormente descritas para el acceso global por parte de cualquier módulo de visualización y es el responsable de avisar al resto de módulos de cualquier cambio en la *ASD_Table*.

Es el responsable de atender las alarmas generadas por el resto de módulos del sistema cuando no existe ningún módulo de visualización que pueda atenderlas. Las almacena en su repositorio interno, activando las acciones y procesos delegados asociados a cada alarma.

En este módulo residen también las rutinas que implican colaboración o cooperación entre varios procesos delegados del sistema, como pueden ser los algoritmos de correlación de alarmas o los módulos de establecimiento dinámico de políticas de gestión basadas en reglas.

Contiene también el repositorio general de alarmas del sistema, de modo que todos los módulos que tienen que almacenar alarmas por períodos largos de tiempo las envían a este módulo. Es recomendable por tanto

3.5.2 Gestor de Adaptadores (MAR).

Este módulo va a permitir, mediante la creación de procesos delegados especializados, enriquecer la información de las alarmas. Todas las alarmas deben ser trasladadas a un formato único (X.733) para facilitar la operación global del sistema. Muchas veces, es necesario enriquecer esas alarmas, entendiendo por esto, añadir información a la alarma que facilite su entendimiento por el operador, dado que la información que generan los elementos de red es, en la mayoría de los casos, muy pobre.

Por tanto, este módulo contiene las rutinas que permiten establecer los puentes entre distintos formatos de alarmas y el formato X.733. Mediante la construcción de estos adaptadores, será posible integrar alarmas de cualquier protocolo en el sistema. Estos adaptadores actúan como gestores de alarmas locales a los agentes estándar o propietarios instalados para tal efecto. Además, cada uno de estos adaptadores cuenta con información estática (almacenada en una base de datos interna) acerca de la construcción de la estructura X.733. De este modo, es posible rellenar de información desconocida (como p.e. causa probable) alarmas provenientes de equipos cuyas alarmas que no disponen de esta información. La información almacenada en la base de datos interna puede provenir de la experiencia del operador en la gestión de un determinado elemento de red, información que puede encontrarse en las especificaciones del equipo pero no en las alarmas, experiencia del operador con determinados tipos de alarmas, etc. Esto nos va a permitir, generalmente, determinar la gravedad y la causa probable de la alarma.

Como se aprecia en la figura 3.10, los procesos delegados almacenados en este módulo capturan las alarmas enviadas por el agente o por un módulo interrogador, las rellenan con información estática y las envían hacia otro proceso delegado.

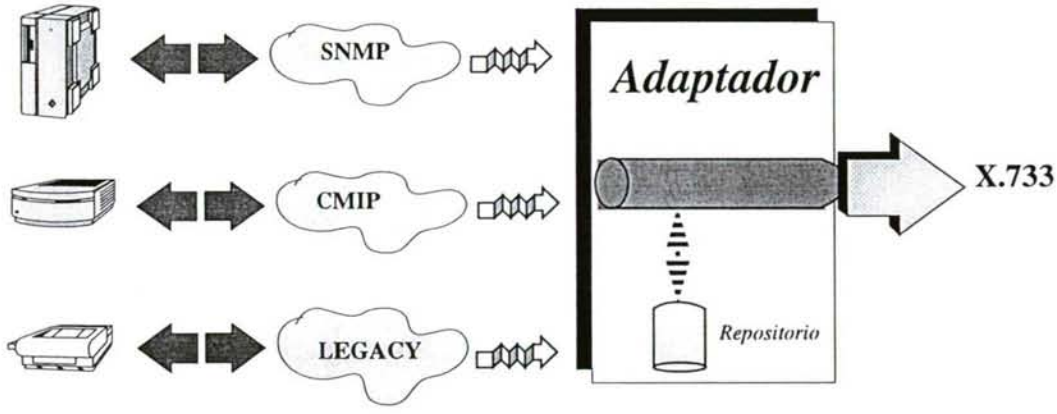

Figura 3.10. *Ejemplos de procesos adaptadores.*

3.5.3 Gestor de Interrogadores (MIR).

En este módulo residen los procesos delegados que implementan las rutinas de *polling*, filtrado, redirección y las que actúan sobre conjuntos o grupos de alarmas (como p.e. correlación).

Las rutinas de *polling* se sitúan entre el adaptador y el agente o *driver* del dispositivo para interrogar periódicamente al mismo, sobre el estado de ciertos componentes del dispositivo. La transición de estados desencadena alarmas que son emitidas hacia el módulo adaptador. En otros casos, se establecen periodos predeterminados de consulta para forzar la emisión de alarmas.

Las rutinas de filtrado se actúan sobre cualquier campo o agrupación de campos del formato de alarmas X.733 [ITU-T X733]. Se sitúan por tanto, después del proceso adaptador. Son delegadas al ASD en el que se encuentra el módulo adaptador del dispositivo y reducen considerablemente el tráfico en la red, dado que únicamente circula por la misma, aquellas alarmas que han superado un determinado filtro.

Las rutinas de redirección permiten enviar alarmas hacia otros módulos distintos al receptor de las mismas. Suelen instalarse en los visualizadores de alarmas con la finalidad de distribuir jerárquica o funcionalmente la gestión de las alarmas que recibe un operador.

3.5.4 Visualizador de alarmas (MVA).

Este módulo constituye la interfaz de usuario con el operador u operadores que están gestionando la red. Puede existir más de un visualizadores de alarmas simultáneamente en el sistema. Puede ser ejecutado como un proceso elástico más, aunque tiene la peculiaridad de acceder directamente al gestor de configuración para, p.e., mantener la consistencia de las tablas del sistema. Algunas de las facilidades que soporta son las siguientes:

- *Control de acceso*: Accede al módulo de configuración para mantener una tabla de usuarios y grupos que determinan los permisos de manejo de los diversos procesos delegados existentes en el sistema.
- *Visualización de alarmas*: Permite visualizar alarmas en formato X.733, ofreciendo al gestor un entorno de operación flexible y configurable, dado que puede seleccionar el número de alarmas y la cantidad de campos que desea recibir. Este módulo recibe también alarmas internas al funcionamiento del sistema, como fallos en la instanciación de un proceso delegado, imposibilidad de cambio de canal de salida, etc.
- *Control de procesos delegados*: Permite acceder a la información de la *PD_Table* global para conocer el estado y localización de todos los procesos delegados del sistema. Facilita al operador un entorno de control interactivo. Desde una ventana de detalle, el usuario puede parar, suspender, reinicializar, terminar o cambiar el canal de salida de cualquiera de los procesos delegados que se están ejecutando (siempre que tenga permisos). También puede instanciar nuevos procesos delegados, teniendo acceso a las tablas de repositorio conocidas, es decir, la de los módulos de configuración, la de interrogadores y la de los adaptadores. Otra de las operaciones que permite el visualizador de alarmas es la posibilidad de modificar los permisos de una determinada instancia, para ello accede al campo permisos de la *PD_Table*.
- *Configuración*: Permite salvar la configuración de trabajo para distintos usuarios del sistema, de forma que cuando un usuario entre en el sistema se encuentre, no sólo la misma configuración del módulo de visualización, sino también, la misma configuración de procesos en el sistema. Para esta última funcionalidad accede a una copia de la *PD_Table* que es mantenida por el módulo de configuración para cada uno de los usuarios en el sistema.
- *Desarrollo*: Tiene acceso a las herramientas que facilitan el desarrollo y almacenamiento de nuevos códigos de procesos delegados en el sistema.

3.5.5 Trabajos relacionados.

En la bibliografía pueden encontrarse muchas referencias para la distribución dinámica de funcionalidad. Entre los métodos más utilizados podemos citar:

- *Remote Procedure Call* (RPC [RPC, 84]): Es el mecanismo de programación de aplicaciones distribuidas más popular. El *RPC Server* debe conocer y tener codificado de antemano el rango de servicios que pueden ser invocados síncronamente por parte de los clientes. Durante la ejecución del procedimiento remoto se bloquea la ejecución del cliente. Sin embargo, en ocasiones no es posible determinar todos los escenarios en los que se va a desarrollar la operación de la aplicación distribuida. Además, la propia iteracción entre cliente y servidor provoca un aumento del volumen de datos intercambiados entre los procesos implicados. El RPC asíncrono (A-RPC) soluciona parte de los problemas de bloqueo. Un ejemplo de diseño de A-RPC lo podemos encontrar en [Liskov, 88].
- *Remote Execution*: Mediante esta técnica se permite a los usuarios ejecutar procesos en *hosts* remotos. Presenta la limitación de que el proceso que vaya a ser ejecutado remotamente debe residir en el *host* remoto previamente. Existen implementaciones basadas en este modelo que permiten la transparencia de localización de los procesos remotos. Otras implementaciones facilitan la creación remota de procesos (siempre que el código esté en el *host* remoto).
- *Remote Evaluation*: Esta técnica permite transferir expresiones de programa entre *hosts* y ser evaluadas remotamente. De esta forma se combina delegación e invocación remota. El procedimiento general es que, un intérprete remoto evalúe la expresión y devuelva al cliente el resultado. Esta iteración es síncrona y el cliente no tiene control sobre la ejecución remota, por lo que puede quedar bloqueado en determinadas situaciones. Además, la implementación de este tipo de distribución es dependiente de un lenguaje y de una máquina, es decir, los programas sólo pueden ser intercambiados entre dos computadores de la misma arquitectura con el mismo entorno de ejecución.
- *Remote Scripting*: Las recientes propuestas en agentes móviles están basadas en lenguajes interpretados, como Telescript [White, 94], en estos escenarios el agente es un *script* que es planificado y ejecutado por un intérprete remoto. El principal inconveniente de estos modelos es su dependencia con respecto a un lenguaje. Frente a esto, los agentes delegados pueden ser codificados en un lenguaje arbitrario, compilado o interpretado. Se consigue un entorno genérico, independiente del lenguaje para extender dinámicamente los procesos bajo control remoto de los mismos. Además, los modelos basados en *scripts* presentan los problemas de dependencia de la funcionalidad del lenguaje interpretado y la imposibilidad de acceder a facilidades de la capa física.

3.6 Conclusiones

Brevemente, se van a esbozar en esta sección las ventajas principales que se derivan de esta arquitectura, en contraposición con las arquitecturas tradicionales de Gestión de Red.

La primera es el elevado grado de distribución del procesamiento, inexistente en la mayoría de los sistemas clásicos. Aquí, los nodos de gestión (ASD) poseen todas las funcionalidades necesarias para realizar tareas efectivas de gestión, incluso sin conectividad. Además, se libera a un nodo central de la carga de procesamiento del sistema completo y de mucho tráfico de gestión (no todo tiene que llegar a él, lo que redunda en mayor rendimiento).

Por otro lado, el modelo está pensado para poder integrar alarmas y eventos de múltiples protocolos (SNMP, CMIP, exclusivos de un dispositivo, etc.), dado que basta con tener PDi especializados en la recepción / emisión acorde con las características de cada uno de estos protocolos para poderlos integrar en el sistema. El uso de un formato estándar y una base de datos simplifican enormemente el desarrollo de aplicaciones de gestión integradoras.

Cuando uno o varios de los ASD, o el gestor de configuración, pierden la conexión, el sistema no deja de trabajar necesariamente, puesto que podemos tener en cada ASD un módulo de control y una serie de procesos delegado que tomen el control en estos casos. Almacenarán la información que se estime oportuna durante el periodo de aislamiento, siendo esta transmitida al GUI del administrador cuando la conectividad se recupere, o accediendo al ASD en local. Esta lógica de subsistencia del ASD es además altamente configurable y flexible.

Para que el sistema disponga de más lógica de proceso, basta con añadir nuevas clases de procesos delegados al repositorio de uno o varios ASD especializados. Teniendo en cuenta que, en general, estos PD son relativamente simples, en relación con los gestores, resulta muy simple añadir lógica para la gestión al sistema. Basta con acceder al repositorio, sin modificar la arquitectura para nada.

En la actualidad, existen numerosos sistemas de gestión en los que también es posible la carga de comportamiento en tiempo de ejecución. Sin embargo, estos sistemas están basados en *Scripts*, por lo que, una vez iniciados, es imposible tener control sobre su ejecución hasta que esta termina por sí misma. En un sistema desarrollado siguiendo la arquitectura propuesta en este trabajo, tenemos control en todo momento de la ejecución de los procesos delegados.

4 .- Implementación de la arquitectura

En este capítulo se presenta la implementación de la arquitectura descrita anteriormente. Para este desarrollo se ha simplificado ligeramente la arquitectura, pe. no se ha incluido el gestor de adaptadores ni el gestor de interrogadores. Tampoco se ha incluido la gestión de usuarios por parte del visor de alarmas. Además sólo se permite un módulo de control por cada proceso elástico. Todas estas modificaciones han sido introducidas para simplificar la implementación y centrarse únicamente en el desarrollo de aquellas partes del sistema que presentan mayor dificultad. Como el lector podrá apreciar posteriormente, la inclusión de estas funcionalidades es sencilla con el desarrollo que se ha llevado a cabo.

Para esta implementación se ha utilizado el lenguaje Java con sus funcionalidades *Remote Method Invocation* (RMI), para la invocación remota y *Java-Database Conectivity* (JDBC) para el acceso transparente a gestores de bases de datos. A pesar de la dependencia con respecto al lenguaje que se introduce este desarrollo, podremos apreciar que la utilización de Java potencia y mejora alguna de las funcionalidades que ofrece la arquitectura propuesta.

4.1 El lenguaje de desarrollo

Como he citado anteriormente, para la implementación del modelo conceptual citado en el capítulo anterior se ha utilizado el lenguaje de programación Java. Actualmente, el lenguaje de programación Java, desarrollado por *SUN Microsystems* [JAVA, 95], ha alcanzado una gran popularidad, debido fundamentalmente a su característica más conocida: *Write Once, Run Anywhere*. Sin embargo, este lenguaje posee otras características que lo hacen particularmente interesante en ciertos ámbitos:

- *Write Once, Run Anywhere*: Esta es la característica más ampliamente reconocida de este entorno: la posibilidad de escribir programas independientes de la plataforma en la que se vayan a ejecutar. Esta posibilidad permite que se reduzca drásticamente el tiempo de desarrollo de sistemas pensados para ejecutarse en múltiples entornos. Como pilar funda-

mental de esta característica nos encontramos con el hecho de que Java es un lenguaje interpretado: un programa Java se ejecuta sobre la *Máquina Virtual de Java*, que sí es dependiente de la plataforma. La figura 4.1 muestra esta característica.

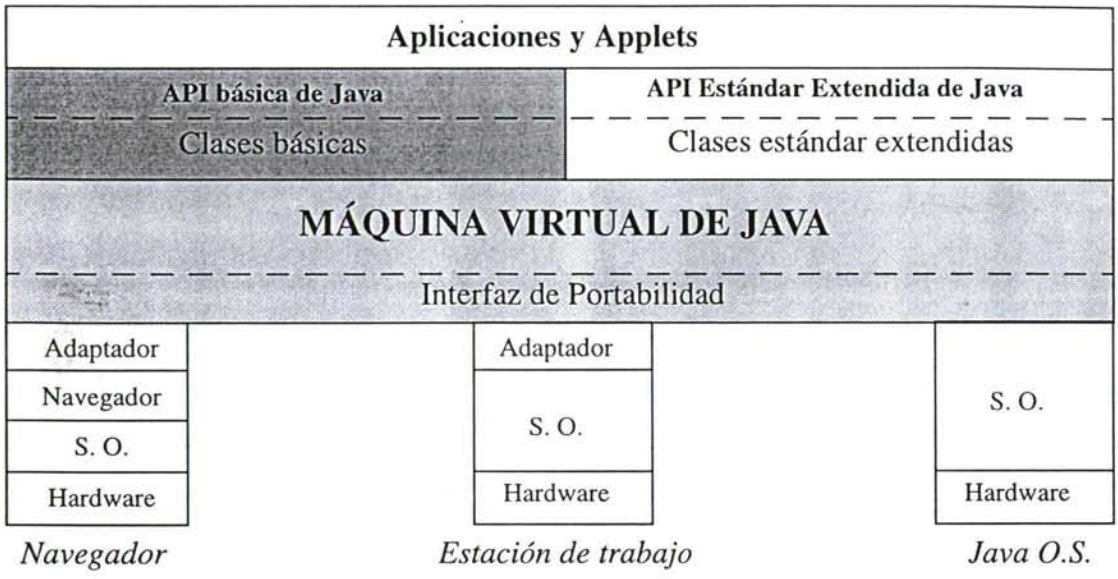

Figura 4.1. *Entorno de ejecución Java*

- *Sencillo y de propósito general*: Para que un desarrollador se sienta satisfecho con un lenguaje de programación, este ha de ser lo más simple posible. En esta línea, Java ha procurado, aún a costa de un cierto decremento en el rendimiento, resultar lo más simple posible de cara al programador. Para ello se han cuidado características como el manejo de excepciones, la gestión de memoria, etc.
- *Totalmente Orientado a Objetos*: Si queremos que nuestros programas puedan cumplir los requerimientos de un desarrollo moderno, con especial atención en la reusabilidad de nuestros componentes software y la compartimentalización de nuestros bloques, la mejor opción es un lenguaje *Orientado a Objetos,* como es el caso de Java.
- *Rendimiento*: Muchas son las críticas que recibe Java en lo que al rendimiento se refiere. Si bien es cierto que su rendimiento, fundamentalmente derivado del hecho de que es un lenguaje interpretado, no es comparable al de otros lenguajes como C++, es suficiente para la mayoría de las aplicaciones. Aunque no podamos desarrollar módulos críticos de control de tiempo real de grandes flujos de datos, sí podemos realizar la mayoría del software (programas de usuario, agentes, etc).
- *Applets*: Los *applets* son una característica interesante del lenguaje. Permiten que un navegador, capacitado para interpretar lenguaje Java -la mayoría del mercado lo están- ejecutar programas escritos en Java.

4.1.1 Características especiales de Java.

Java nos ofrece la posibilidad de cubrir de un modo sencillo aspectos que van desde la movilidad del comportamiento entre los distintos agentes del sistema hasta la posibilidad de una gestión cómoda, rápida y fácil de mantener de las comunicaciones, incluyendo un mecanismo de *RPCs* en

la *API* básica del lenguaje. A continuación se detallan estas características, junto con alguna más relacionada:

- *Movilidad, serialización y aspectos relacionados*: Dado que Java es un lenguaje interpretado, resulta sencillo hacer que las clases que componen un programa Java puedan moverse en tiempo de ejecución. Para ello, el entorno de Java viene provisto de varios *cargadores de clases*, que se encargan de obtener el comportamiento de una clase en tiempo de ejecución. Estos cargadores pueden obtener las clases tanto del disco como de la red, posibilitando así la carga de comportamiento a través de la red. Este último es el caso, por ejemplo, de los *Applets*, cuyo comportamiento se carga de una *intranet* o de *Internet* dinámicamente, en tiempo de ejecución. Además, no sólo las clases pueden ser cargadas en tiempo de ejecución, sino también los objetos o instancias de las mismas. El mecanismo que posibilita esto es la serialización, y también está incluido en la especificación básica del entorno Java. Por medio de características como la carga dinámica de clases o la serilización podemos realizar tareas como cargar determinada lógica, susceptible de ser actualizada frecuentemente, de un servidor, para después crear instancias con esa lógica. Además, resulta sencillo el crear objetos persistentes, por medio de la serialización, de tal forma que estos no pierdan su validez al acabar la ejecución del programa. Esto es muy útil si se desea crear un repositorio de objetos móviles.
- *RMI* (*Remote Method Invocation* [RMI, 96]): Si se pretende realizar software con unas ciertas capacidades en cuanto a comunicaciones, salvo que los criterios de rendimiento sean especialmente exigentes o que se desee realizar un cliente o un servidor para un protocolo ya existente, la utilización de un esquema de *RPC* resulta muy ventajosa. En Java, se dispone de RMI, que soporta un esquema de RPC orientado a objetos, con soporte de todas las funcionalidades propias del lenguaje, pues forma parte inseparable del mismo. Por medio de este mecanismo, aparte de la sencillez con la que dos agentes intercambian información, encapsulándola con la misma sintaxis que una invocación a un método, permite automatizar todo el proceso de paso de objetos y la carga remota de los mismos, en caso de no encontrarse disponible la versión de la clase a la que pertenecen (o no encontrarse ninguna versión). De este modo, podemos aprovechar las características anteriores (serialización y carga remota de clases) de forma simple y automática. RMI va a permitir ver todo un sistema distribuido como un único proceso, a nivel de complejidad sintáctica, aunque seremos conscientes de que ambos se encuentran en diferentes máquinas virtuales (independientemente de si están en la misma máquina o no).
- *Comunicaciones básicas*: Para la creación de sistemas que soporten protocolos existentes, Java nos proporciona una API bastante evolucionada de manejo de *sockets*, del estilo de los que hay disponibles en UNIX. Su uso resulta eficiente y simple. Además, es posible crear nuestra propia *factoría de sockets*; podemos personalizarlos, por ejemplo, para que soporten la encriptación directamente.
- *Seguridad*: Java incorpora características como los Gestores de Seguridad, que restringen los permisos de acceso en las operaciones básicas, en función del tipo de conexión, o soporte de encriptación en la propia API. Además, su diseño es suficientemente abierto como para añadir nuevas librerías de clases que den nuevas opciones de seguridad, modernas y de muy fácil uso. Por citar un ejemplo, las opciones más comunes para encriptación están disponibles en las últimas versiones del kit de desarrollo, y se irán añadiendo más. Otro ejemplo, es la posibilidad de encriptar, próximamente, las conexiones con RMI.

- *Java-Database-Connectivity (JDBC)*: Java nos proporciona una API para acceder a bases de datos de una manera sencilla e intuitiva, JDBC. Podemos pensar en JDBC como en una versión Java de ODBC. La mayoría de las grandes compañías de bases de datos ya han creado sus *drivers* para permitir a Java acceder a sus productos. Existe, sin embargo, el *driver* JDBC-ODBC desarrollado por *JavaSoft*, que permite la comunicación de Java con bases de datos exclusivas. Por lo tanto, Java y JDBC proporcionan una solución muy potente para escribir aplicaciones portables para bases de datos.
- *Movilidad de código*: En el modelo conceptual se hace uso del concepto de movilidad de comportamiento, fundamentalmente para los mecanismos de delegación. Si deseamos desarrollar sistemas con características sencillas de movilidad de comportamiento, Java nos aporta ya en su especificación estándar una característica muy interesante, que es la *carga dinámica de clases*. Junto con la posibilidad de mover objetos a través de la red, por medio de la *serialización*, podemos hacer que nuestros sistemas sean capaces de mover objetos a través de la red, incluso si la clase a la que perteneces no se encuentra disponible en el destino. Otra opción es cargar explícitamente la clase a la que pertenece un objeto. También debemos enviar y recibir explícitamente los objetos serializados.

4.2 Implementación de la arquitectura de soporte

Como se ha citado en el capítulo anterior, existe una serie de módulos, denominados ASDs que contienen toda la lógica de gestión del sistema. Estos nodos están distribuidos por el sistema, cerca (generalmente en la misma máquina o en una adyacente) del dispositivo a gestionar.

Básicamente, las capacidades del ASD, son las de planificación y control de PDIs, repositorio, gestor de órdenes remotas y lógica de gobierno general del ASD. Así mismo, según el tipo de ASD, pueden existir pequeñas diferencias, fundamentalmente en la lógica de gobierno general del ASD y pequeños módulos (paquetes de Java) añadidos.

Con la utilización de Java la estructura en capas del ASD se simplifica. Java nos abstrae de la capa de S.O. (es independiente de la plataforma), tanto en la programación de nuevas funcionalidades por encima del ASD como en la programación básica del mismo. En la figura 4.2 podemos apreciar los módulos más relevantes de la implementación del ASD, junto con sus interrelaciones más importantes.

Los cambios fundamentales son: la supresión de la capa de S.O. (Java es multiplataforma), las funcionalidades del módulo de comunicaciones van embebidas en otros módulos y los módulos principal y de control van integrados con el planificador.

Tal y como se ha mencionado anteriormente, el ASD está estructurado en dos capas fundamentales, en esta implementación, ya que los servicios del S.O. no son accesibles directamente por el programador en Java. Las capas de que consta pues el ASD son la de *Aplicación*, en la que se ejecutan los PDIs dando al sistema su comportamiento visible de cara al exterior (la lógica que realiza finalmente la gestión) y la capa de *Backplane*, en la que el sistema soporta toda la lógica básica del ASD.

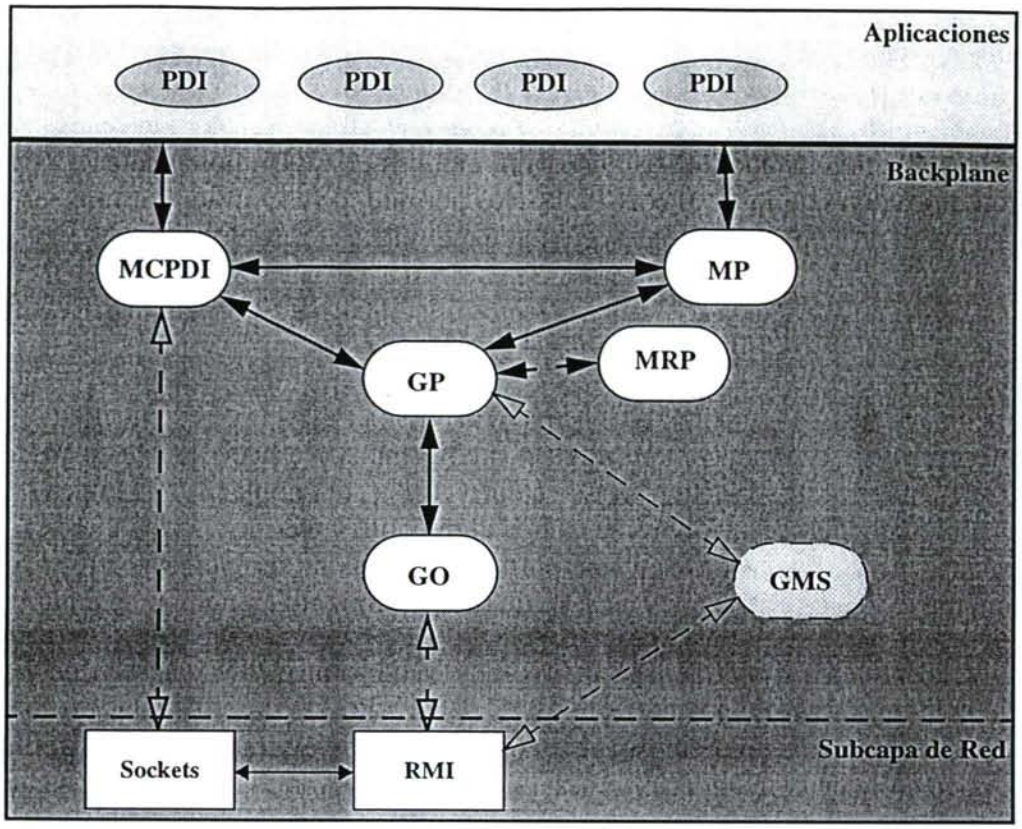

Figura 4.2. *Estructura interna del ASD*

Mención especial merece la subcapa de *Red*, dentro de la de *Backplane*, puesto que esta tiene dos funcionalidades fundamentales. La primera es proporcionar al programador un mecanismo de *sockets* para el acceso a la red (TCP/IP) y la segunda un mecanismo de RPCs, integrado en Java, denominado RMI. Con el acceso a estos dos servicios de Java, los diversos módulos que requieren servicios de comunicación con otras máquinas pueden ser atendidos.

A continuación se describe cada uno de los módulos que constituyen la estructura general del ASD.

4.2.1 Gestor Principal (GP).

Es el módulo central del ASD y tiene la responsabilidad de coordinar al resto de módulos, haciendo, en la mayoría de las ocasiones, de entidad intermedia que decide a qué módulos hay que involucrar en el despacho de un servicio determinado.

4.2.2 Módulo RePositorio (MRP).

Se encarga de gestionar los Procesos Delegables (PD) de que dispone, tanto para crear instancias locales como para ser delegados a otros ASD (o incluso como almacén de procesos recibidos de otros ASD del sistema). Sus servicios son los típicos de un repositorio (consultar si está uno concreto, grabar, cargar, etc.).

4.2.3 Módulo Planificador (MP).

Gestiona los *threads* de los PDI que estén instanciados. Cuando se inicia un nuevo PDI en el ASD, este es ejecutado como un *thread* (uno para cada PDI, aunque podrían ser varios, por ejemplo uno para cada fuente de entrada). Permite suspender, reanudar o terminar la ejecución de las instancias, así como acceder a funcionalidades de las mismas (métodos que estas implementen), a través del Módulo de Comunicación con los PDI (MCPDI). Nos permite, de esta última forma, dar órdenes más concretas a los PDI (cambio de canales, envío de mensajes, etc.).

4.2.4 Módulo de Comunicación con los PDI (MCPDI).

Este módulo tiene acceso a las funcionalidades internas de los PDI que se ejecutan en el entorno de un ASD determinado. Controla, como se expuso anteriormente, los canales de los PDI, el envío de mensajes, espera por eventos, o cualquier otra característica contemplada en la lógica soportada por los PDI. El estado de ejecución recae sobre el módulo anterior. Dado que no se tiene conocimiento, en general, de los PDI en tiempo de compilación, se realiza por medio de una superclase común a todos los PDI, de la que sí se tiene conocimiento en tiempo de compilación.

A más bajo nivel, emplea *sockets* para la intercomunicación entre PDI, ya sean en la misma máquina o en máquinas (virtuales) diferentes. De este modo se simplifica la gestión de las comunicaciones y se uniformizan. Este mecanismo, aunque simple y no tan versátil como el que da RMI, tiene poco coste de comunicaciones -este tipo de mensajería es mucho más frecuente que el paso de órdenes-.

4.2.5 Gestor de Órdenes (GO).

Cuando un ASD desea enviar órdenes (peticiones) a otro ASD en el sistema, los servicios necesarios son proporcionados a través de este módulo. Implementa toda la lógica pertinente para la gestión / emisión de órdenes entre ASD, proporcionando una sintaxis en forma de invocaciones de métodos similares a métodos locales. La API de la que dispone el programador está estructurada en dos niveles de abstracción distintos, de los cuales el inferior depende fuertemente de la localización, pero el superior no depende de la localización. Esta estructuración hace que sea muy simple su uso.

A más bajo nivel, emplea RMI para la invocación remota de métodos. Esta parte del Java emplea sockets o encauzamiento a través de HTTP, en el caso de que exista un *firewall* intermedio.

4.2.6 Gestor de Módulos Superiores (GMS).

Cuando es necesario aumentar las funcionalidades del sistema, los ASD dispondrán de módulos que se ejecutan "*en un nivel superior*", y que se comunican con los mismos por medio de métodos remotos (RMI). El mecanismo de comunicaciones empleado es prácticamente el mismo que en el caso de la gestión de órdenes remotas aunque, evidentemente, con una sintaxis sustancialmente diferente, además de ser esta dependiente del módulo en cuestión.

Este módulo se encargará de soportar las órdenes que procedan de estos módulos, enlazados dinámicamente con el ASD, y de traducirlas a órdenes de nivel inferior (locales o remotas) según proceda. Un ejemplo de este proceder lo constituyen los visualizadores de alarmas (MVA), que se enlaza dinámicamente con el módulo de configuración (MCF), y que es capaz de invocar, entre otras, órdenes que se corresponden con las de la API de nivel superior de control de PDI.

Cada tipo de ASD posee -o no- uno de estos gestores, que soportará un determinado conjunto de interfaces, en función de los tipos de módulos integrados, junto con la lógica necesaria para atender estas órdenes. Opcionalmente, si estos módulos aceptan la invocación de órdenes en sentido inverso, en los módulos existirá un gestor que realizará lo complementario. Veremos posteriormente un ejemplo de integración de un módulo superior en un ASD que aclarará con mayor detalle este aspecto.

4.3 Tipos de ASD implementados.

Para esta implementación se han desarrollado dos tipos diferenciados de ASD ejecutables en el sistema. Uno, es el ASD genérico, que tiene todas las funcionalidades generales de un ASD, pero que no es contenedor de lógica específica de gestión ni tiene funcionalidades especiales. El otro, se corresponde con las funcionalidades adicionales de un gestor de configuración (MCF). Entre sus funcionalidades se encuentra el control de la tabla global de PDI del sistema o la gestión de Módulos de Visualización de Alarmas (MVA).

4.3.1 ASD estándar.

Dentro de un ASD nos encontraremos con los módulos citados anteriormente, lo que nos permitirá acceder a los servicios básicos de los nodos de gestión: repositorio, mecanismos de delegación, manejo de órdenes remotas entre ASD, planificación de PDI (*threads*), y comunicaciones entre PDI. Estas son, básicamente, las funcionalidades que nos ofrece un ASD convencional.

4.3.2 Gestor de configuración (MCF).

Es preciso que en el sistema se esté ejecutando un gestor de configuración, con una dirección de acceso conocida por los demás, que realice ciertas tareas de coordinación de ASDs. Las funcionalidades que debe aportar un ASD de este tipo son: Gestión de tabla global de PDI, gestión de tabla de ASD en el sistema, soporte de funcionalidades del visor de alarmas y repositorio centralizado de procesos delegables (debido a la no incorporación del gestor de adaptadores y del gestor de interrogadores).

En la siguiente figura podemos apreciar cómo se comportan los ASD en el sistema, con especial hincapié en los módulos que componen a cada uno (generales y específicos) y su relación:

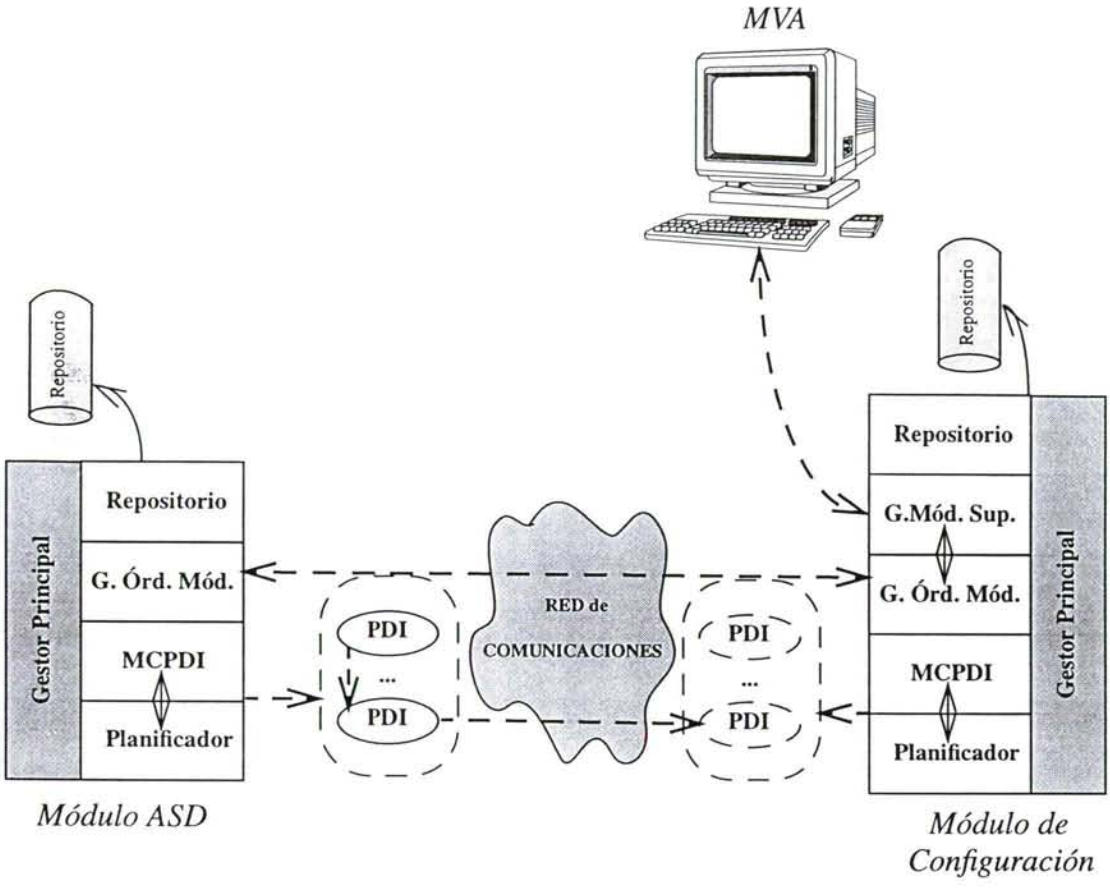

Figura 4.3. *Interconexión entre ASDs*

En el gráfico se puede ver cómo se relacionan un ASD normal y un gestor de configuración, además de un visualizador de alarmas (MVA) enlazado al segundo. Se puede apreciar cómo este enlace se realiza a través del GMS. De la misma forma, en ambos ASDs el MCPDI y el Planificador tienen una estrecha relación. Todos los módulos están gobernados por el GP, que varía según el tipo de ASD del que se trate.

Cuando dos PDI desean comunicarse, la lógica de intercomunicación residente en el MCPDI se encarga de enlazarlos, dinámicamente, a través de *sockets*, bien en local o bien a través de la red. La comunicación por medio de órdenes, como puede verse, es bidireccional, y está controlada por el GO.

4.4 Arquitectura de comunicaciones.

En este apartado se describen los mecanismos básicos y específicos de comunicaciones, tanto entre dos ASDs (en forma de *invocaciones de métodos remotos*) como entre dos PDIs (paso de mensajes directos utilizando Java). A continuación se detallan más ambos apartados:

- *Invocación de primitivas del ASD:*

  El ASD soporta una serie de **primitivas**, las citadas en el capítulo anterior, que permiten, por ejemplo, delegar procesos, solicitarlos, instanciarlos y controlarlos en general, actualizar tablas, etc. En este sistema, estas primitivas están accesibles a otros ASD por medio de *RMI*, como métodos remotos. Este mecanismo proporciona una interfaz cómoda y simple, simplificando los protocolos y la mensajería.

- *Mensajes entre PDI:*

  Los PDI poseen un canal de entrada y un canal de salida; estos canales se emplean para recibir datos, sin intermediarios, de otros PDI, o mandarlos a otros. Para estas comunicaciones, se habilitan dos *sockets*, convenientemente configurados, en cada PDI, que son usados para emitir y recibir, sin intermediarios. Estos canales son asignados y controlados por el gestor principal responsable del PDI.

4.4.1 Métodos de comunicación entre PDI.

Existen dos métodos que pueden ser invocados en los PDI para comunicarse. Los métodos que se implementan son:

- *Mensaje* **recibirMensaje***()*

  Recibe un mensaje, en forma de objeto, cuya interfaz debe ser conocida, que contendrá la información del mensaje (por ejemplo, una *trama X-733*). También se valora la posibilidad de forzarlo a un tipo de clase superior, con los métodos básicos comunes a todos los mensajes.

- **enviarMensaje***(Mensaje)*

  Envía un mensaje para que sea recogido por el método anterior al otro lado. También debe ser de tipo conocido de antemano al otro extremo.

4.4.2 Interrelaciones del módulo de comunicaciones

El modulo de comunicaciones interactúa con varios módulos del sistema, a través de las primitivas que éste proporciona para ser invocadas por otros ASD:

- *Repositorio:*

  Con este módulo, se comunica para almacenar o recuperar los procesos delegados, tanto de memoria como de disco (transparentemente), a través de las primitivas de delegación (ASD receptor) como de solicitud de delegación (solicitante).

- *Gestor de tabla de ASDs (**ASD**):*

  Cuando se invoca una primitiva para actualizar la tabla de ASDs accesibles desde uno dado, el MCM interactúa con este módulo. Este modulo se especifica a continuación, junto con la estructura de sus entradas. Como gestor de esta tabla actuará el módulo ASD.

- *Gestor de tabla de PDI (**Planificador**):*

  En aquellas primitivas que se encarguen de hacer llegar órdenes a una instancia de proceso delegado (*PDI*), el MCM tendrá que interactuar con este módulo para que actualice las entradas en esta tabla del ASD. Este módulo se especifica a continuación, junto con la estructura de sus entradas. Como gestor de esta tabla actuará el Planificador de Procesos Delegados.

4.4.3 Tablas del sistema.

Dentro del sistema se manejan una serie de identificadores, para los procesos delegados (PDI), los módulos que implementan la arquitectura básica (ASD) y los códigos de los procesos que pueden ser delegados (CPD). En adelante, llamaremos a estos identificadores *PDId*, *ASDid* y *CPDId*, respectivamente. Todas las direcciones en el sistema tendrán un formato único compuesto por:

- ASD (Que solicita la delegación): 2 bytes.
- MOC (Módulo remoto del que dependerá): 1 byte.
- CPD: 2 byte ASD en que reside, 2 bytes CPDId global del sistema.
- PDI: 1 byte.

Tanto para el ASDid como para el CPDId se reserva un mayor número de identificadores posibles, pues puede haber, aunque no sea frecuente, muchos ASD o muchos tipos de proceso delegable. En el caso de los PDId no es necesario, pues es imposible que una máquina soporte un número tan grande de procesos delegados.

Cualquier identificador de PDI lleva toda esta información, indicando el ASD - MOC del que depende (que solicitó que le instanciasen), así como el tipo de proceso delegado que es y el identificador de instancia. En aquellos lugares en los que no sea necesaria toda esta información, estas direcciones se pueden truncar.

4.4.3.1 Tabla de ASDs.

En cada ASD del sistema, sea éste del tipo que sea, existe una tabla con información sobre los demás ASD disponibles en el sistema. Esta tabla contendrá siempre todos los ASD accesibles; ninguno tendrá tratamiento diferente. La información contenida para cada entrada de esta tabla es la siguiente:

- *Identificador*: Identificador de 2 bytes del ASD.
- *Tipo de ASD*: Tipo del ASD (MOCA, MVA, etc).
- *Nombre*: Nombre textual que recibe el ASD.
- *Dirección DNS* de la máquina: Localización de la máquina del ASD (DNS).
- *Dirección IP de la máquina*: Localización de la máquina del ASD (IP). O este o el otro son opcionales.
- *Puerto de escucha*: Puerto de escucha del registro (invocación remota de primitivas).

4.4.3.2 Tabla de *proxies* del sistema.

Para poder realizar invocaciones a primitivas de otros ASD del sistema, es necesario mantener referencias a los objetos que estos mantienen accesibles a través de RMI. Una vez que se realiza el enlace entre estos objetos remotos y nuestro ASD, estos *proxies* se almacenan en una tabla, similar a la de los ASD, para tenerlos siempre accesibles.

La convención para la nomenclatura en el registro de los objetos remotos es fija y se describe a continuación:

- ASD + ASDid + tipo ASD

Esta tabla se mantiene mediante el uso de primitivas de actualización de las tablas de ASD. En caso de que el *proxy* de un ASD no sea accesible, su correspondiente entrada en la tabla de información de ASD se eliminará.

4.4.3.3 Tabla de PDIs.

Además de la tabla de ASDs disponibles en el sistema, cada ASD tiene una tabla que contiene los PDI que actualmente están trabajando dentro de su entorno de ejecución. La información contenida en cada entrada de esa tabla de PDI del sistema es la siguiente:

- *Identificador*: Identificador completo del PDI (ASDid, MOCid, CPDId, IdInstancia).
- *Descripción*: Cadena textual describiendo al PDI -para el usuario-.
- *Propietario:* Usuario que ha instanciado el proceso delegado.
- *Grupo:* Grupo al que pertenece el usuario propietario de la instancia.
- *Permisos:* Permisos de que dispone el propietario, grupo y resto de usuarios con respecto a las operaciones que pueden llevarse a cabo sobre la instancia.
- *Estado*: Estado de ejecución de la instancia (Suspendida, Terminada, ...).
- *ASD que lo lanzó*: ASD que ordena la instanciación de este PDI.
- *ASD que lo hospeda*: ASD en el que se encuentra el PDI. Importante si el PDI es nuestro pero no está en nuestro entorno de ejecución.
- *MOC responsable*: MOC responsable de su instanciación dentro del ASD anterior.
- *Canal de entrada*: Puerto por el que esta instancia se mantiene a la escucha.
- *Canal de salida*: Dirección y puerto por donde esta instancia emite mensajes.

Es importante destacar que los campos de ASD responsable y ASD huésped se usan para poder controlar los permisos de acceso a los PDI de forma efectiva. Por ejemplo, si un ASD solicita a otro que le instancie un PDI, sólo aceptará órdenes para ese PDI que provengan de quien ordenó su instanciación.

Para que un tercer ASD pueda enviar órdenes a un ASD, éstas tendrán que pasar por el ASD que ordenó su instanciación (de este modo se controlan perfecta y fácilmente las restricciones de acceso). Éste, si acepta la petición, enviará la orden solicitada al huésped del PDI, que ejecutará la orden finalmente.

4.4.4 Coherencia de tablas en el sistema.

Cada uno de los ASD que están en el sistema posee, como se mencionó anteriormente, una tabla en la que almacena la información de los PDI a los que está dando servicio, además de los PDI de los que es responsable. Además de estas tablas, una por cada ASD, existe otra, en el gestor de configuración (MCF) que tiene una entrada, análoga a las de las demás tablas, en la que almacena la información de todos y cada uno de los PDI que se están ejecutando en el sistema.

A tales efectos, todos los ASD del sistema deben notificar todo cambio en sus tablas de PDI al gestor de configuración. Para ello existen dos primitivas de actualización de tabla global, una para añadir / modificar una entrada y otra para eliminar una entrada.

Toda la lógica de actualización de las tablas se encuentra encapsulada en la clase que se encarga del manejo específico de estas tablas (*HashtablePDI*). De esta forma, cuando se produce una actualización en una tabla, el programador queda liberado de la tarea de tener que encargarse de mantener la coherencia de todas las tablas del sistema; esto último acarrearía problemas, debidos a la dispersión de la lógica encargada de la delicada tarea de mantener esta coherencia y a la replicación de la misma. Este es, pues, un mecanismo delicado que, si no se contempla adecuadamente, puede acarrear múltiples problemas de descoordinación entre los ASD del sistema.

Cuando se produce un cambio en la tabla global, este es notificado a los visualizadores de alarmas, en el supuesto de que se encuentren en estado de ejecución. Si no, simplemente se almacena, pasándosele, al visualizador de alarmas, la tabla entera cuando se inicialice. De este modo, siempre podemos mantener una visión global del sistema.

4.5 API de órdenes

Amparándose en las primitivas de comunicaciones, cualquier ASD va a ofrecer al programador una serie de funcionalidades de más alto nivel, que se encarguen de ocultar las vicisitudes internas del sistema, fundamentalmente las referidas a la localización de los destinatarios de los ASD, permisos y secuencia de operaciones básicas. A tales efectos, se define una *API* de más alto nivel que enmascara la complejidad del uso de las primitivas del sistema.

Esta *API* se encargará de invocar a la/s primitiva/s necesaria/s en cada caso, así como seleccionar sus parámetros y escoger el *proxy* adecuado en cada caso. Las funciones que va a soportar esta *API* son:

Instrucción	Descripción
PDId **Instanciar_PD(** *ASDidDest,* *CPDId,* *descripción*)	Crea una instancia de un Proceso Delegado, en el ASD que especifica. Cuando se invoca esta sentencia, el sistema debe comprobar si el código de la clase a la que pertenece la instancia que se desea crear existe. En caso contrario, se procedería a intentar delegarlo. La instanciación se realiza con *SDR_Instantiate()*

Suspender_PD(*PDId*)	Suspende la ejecución de una instancia. Simplemente, se encarga de hacer llegar una primitiva *SDR_Suspend()* al ASD responsable de la instancia implicada.
Reanudar_PD(*PDId*)	Reanuda la ejecución de una instancia. Se encarga de hacer llegar una primitiva *SDR_Resume()* al ASD responsable de la instancia implicada.
Terminar_PD(*PDId*)	Finaliza la ejecución de una instancia. Hará llegar una primitiva *SDR_Terminate()* al ASD responsable del PDI. En caso de que ya no haya ninguna instancia de ese código, se invocará a *SDR_Delete()* en ASD que hospedaba a la instancia.
PonerCanalSalida_PD(*PDIdOrig,* *PDIdDest*)	Conecta la salida de una instancia a la entrada de otra, para que puedan procesar la información en secuencia. Comprobará cuál es el canal de entrada para el PDI de destino. Una vez hecha esta comprobación, se encargará de fijar el canal de salida del PDI origen con *SDR_SetChannelOut()*.

Como se puede apreciar, el conjunto de métodos disponibles para el programador se simplifica enormemente, sin menoscabo de funcionalidad alguna en el sistema. Todas aquellas funciones que podría, a priori, parecer que se eliminan son en realidad empleadas internamente por estos métodos de más alto nivel.

Por ejemplo, las primitivas de delegación y borrado del repositorio quedan supeditadas a la orden de instanciación -se generan primitivas de delegación en caso de que no se encuentre el código en el ASD de destino- o a la orden de terminación -se genera una primitiva de borrado si ya no quedan instancias de ese código-. En el caso de las primitivas de manejo de tablas de ASD, se usan en el código básico del ASD, nunca por parte del programador.

De igual modo, las primitivas de petición de canal de entrada y de información de canal de entrada de un PDI no se usan directamente -a cada PDI se le fija un canal de entrada automáticamente y la segunda se llama para especificar el canal de salida-.

4.5.1 Traducción a invocaciones remotas.

Las órdenes de alto nivel anteriores pueden ser invocadas bien en local, dentro de un ASD o bien desde algún módulo anexo a un ASD, como por ejemplo el *Módulo de Visualización* (MVA). En el segundo de los casos, estas órdenes de alto nivel tendrán una correspondencia en remoto, que se limitará, básicamente, a invocar a sus correspondientes métodos locales en el ASD.

4.5.1.1 Un ejemplo de interrelación entre ASDs

A modo de esquema de interacción, se puede representar toda la interacción, en lo referente a las

comunicaciones, entre el visor de alarmas y los ASD del sistema En el siguiente esquema se pretende resaltar la secuencia de pasos a dar para una orden desde el visor de alarmas, y la secuencia de pasos que se da cuando se le notifica algún cambio en algún PDI, para que este lo refleje de cara al usuario.

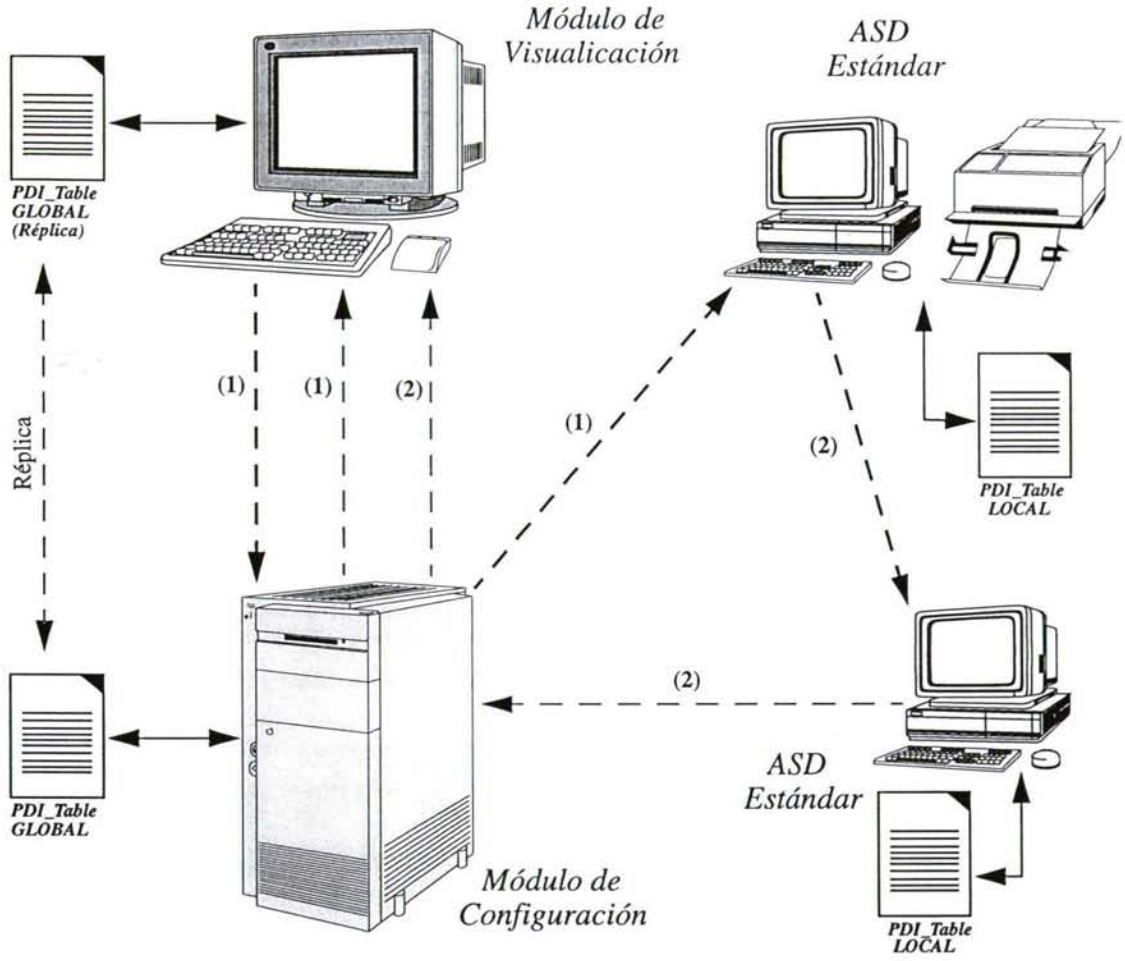

Figura 4.4. *Interacción entre el visor de alarmas y el resto de módulos.*

En el gráfico anterior se refleja el tránsito de las operaciones relevantes en una situación típica. Para ilustrar la interacción entre el visor de alarmas, el gestor de configuración y los demás ASD del sistema, se representa un escenario en el que hay dos órdenes, una emitida desde el visor de alarmas y otra desde un ASD cualquiera del sistema a otro, haciendo especial hincapié en las interrelaciones entre los módulos. Las operaciones son, por ejemplo:

1. *Instanciación en un proceso delegado desde el visor de alarmas.*

   En este caso, el usuario pretende instanciar un PDI en un ASD. La orden llega al gestor de configuración, que la dirige al destinatario correcto -suponiendo que no involucre a un tercer ASD-. Por otro lado, esta orden genera una actualización en las tablas de PDI local -ASD- y global -gestor de configuración y visor de alarmas-. Ambos flujos están reflejados con la etiqueta **(1)**.

2.*Suspensión en un proceso delegado desde otro ASD.*

Un ASD cualquiera del sistema, responsable de un PDI en otro ASD, ordena a este que suspenda su ejecución. Esto origina una actualización en las tablas de PDI locales -ambos ASD- y global -gestor de configuración y visor de alarmas-. La secuencia de actualizaciones se puede ver con la etiqueta **(2)**.

Como se puede observar, el mecanismo resulta simple y, a la vez, versátil, pues permite monitorizar, si el usuario dispone de los permisos adecuados, todos los PDI del sistema. Se optó por una gestión -que no capacidad de control- centralizada en el gestor de configuración por sencillez, basándose en el hecho de que este tipo de cambios (muy relacionado con las primitivas de los ASD) no van, en principio, a generar un tráfico excesivamente elevado, ni siquiera en situaciones críticas, donde más daño podría hacer.

Como alternativa, y una vez que se haya probado y, preferentemente, cuantificado, el tráfico que esta política de gestión supone, se podría valorar la posibilidad de optar por una gestión distribuida de las tablas de PDI, donde el/los visores de alarmas del sistema se "*registren*" a unos determinados PDI y sólo reciban notificación de esos. Con este otro esquema, se reduce el tráfico -por determinar cuánto- pero se aumenta bastante la complejidad lógica.

4.6 Integración de módulos

La flexibilidad es un factor clave, por lo que se debe cuidar en extremo. En este sentido, y persiguiendo el mayor grado de flexibilidad posible, se ha diseñado el sistema para que soporte un amplio abanico de módulos de nivel superior a los ASD. En pocas palabras, es posible que se añada, con suma facilidad y sin alterar ningún punto de importancia en el diseño de los ASD, un módulo con funcionalidades específicas, autónomo y distribuible, que dé servicios adicionales al sistema.

Para explicar esto de un modo simple, baste mencionar un módulo de este tipo que se ha desarrollado: el *Módulo de Visualización de Alarmas* (MVA). Este módulo se relaciona con el gestor de configuración, permitiéndonos tener una visión global de los PDI que se están ejecutando en el sistema, además de instanciar otros nuevos o realizar operaciones de control sobre los mismos y, fundamentalmente, visualizar las alarmas que se producen en el sistema.

A pesar de ser este un módulo primordial para el aprovechamiento real del sistema, este módulo se ejecuta aparte de los ASD, relacionándose con el gestor de configuración a través de la red, como componente aparte del sistema. Esta filosofía, a la vez que simple, resulta versátil, pues es el puente para facilitar la apertura del sistema a nuevas funcionalidades. A continuación se explica la base de esta integración, con especial mención a la integración del gestor de configuración con el visor de alarmas.

4.6.1 Esquema básico de la interrelación de los ASD - Módulo Superior.

Cuando se desea enlazar uno de estos módulos con un ASD, la lógica que se necesita, básicamente,

es la siguiente:

- Interfaz Módulo - ASD

  Esta interfaz, obligatoria en cualquier caso, es la que nos va a permitir que el módulo enlazado realice invocaciones a métodos soportados, remotamente, por el ASD del que depende. Mediante estos métodos, el módulo podrá realizar peticiones de servicios al ASD, y el conocimiento que el módulo tiene que tener del ASD queda reducido a esta interfaz y a los tipos de los parámetros que se manden o reciban como resultado en sus métodos. La interfaz módulo - ASD debe estar implementada, pues, en el ASD, que será quien realice el cómputo de la operación en ella solicitada.

- Interfaz ASD - Módulo. Opcional

  En aquellos módulos que, además de enviar peticiones a su ASD, necesiten de la posibilidad de proporcionar servicios bajo demanda del ASD (por ejemplo, ser notificados de algún evento del que tenga conocimiento el ASD), debe haber una interfaz de este tipo. Esta interfaz soportará todos aquellos métodos que deban ser accesibles desde el ASD, además de ser conocida en el ASD -va a tener una referencia remota que implementa esta interfaz-. Los métodos de esta interfaz, análogamente al caso anterior, deberán ser implementados en donde se van a atender las llamadas; es decir, en este caso en el módulo de nivel superior.

- Gestor de conexión Módulo - ASD (en el ASD)

  Aquí se realizarán las tareas de registro del gestor de conexión -en el ASD- para recibir peticiones de módulos superiores y despacho de las invocaciones remotas de métodos desde los mismos. Básicamente, realiza lo mismo que el *Gestor de Primitivas*. Este gestor de conexión soportará tantas interfaces remotas como tipos de módulo adicional soporte, suponiendo que cada tipo necesite una interfaz deferente de las demás.

- Gestor de conexión ASD - Módulo (en el módulo). Opcional

  Este gestor sólo será necesario en el caso de que el módulo superior también soporte invocaciones a métodos desde el ASD al que se enlazó. En este caso, aparte de hacer el *lookup* del Gestor de conexión Módulo - ASD, deberá notificar su existencia al ASD y darle una referencia remota de un objeto exportado que implemente su interfaz remota. De este modo, el ASD estará capacitado para realizar invocaciones a métodos en el módulo. Tanto si este gestor existe como si no, es preciso que el módulo haga un *lookup* al registro del ASD, para obtener su *proxy* y poder iniciar la cooperación.

Para conseguir que estos módulos se comporten como si realmente estuviesen en la misma máquina virtual se emplea el mismo mecanismo que en el paso de órdenes remotas entre los ASD:

RMI. Gracias a este mecanismo, podemos realizar la conexión entre los módulos y los ASD, bien unidireccional (métodos del módulo al ASD) o bidireccional, de un modo sencillo, intuitivo para el programador y fácil de modificar.

4.6.2 Inicialización de los módulos.

La secuencia de pasos para permitir que uno de estos módulos pueda cooperar con un ASD es bastante simple. En esencia, es preciso que el ASD esté trabajando (al iniciarse ya crea el enlace en el registro para los módulos). Cuando un ASD se incializa, automáticamente se lanza la escucha para la conexión de módulos, es decir: el Gestor de Conexión Módulo - ASD.

El siguiente paso será la búsqueda, o *look up*, por parte del módulo de nivel superior. Con esto, el módulo estará en disposición de poder realizar invocaciones al ASD, por medio del *proxy* que le devuelve la instrucción de *look up*. Si, además de las invocaciones desde el módulo al ASD, se permiten invocaciones en sentido contrario, se utilizará una orden para notificar la presencia de un nuevo módulo conectado al ASD, además de proporcionarle al mismo un *proxy* del módulo. Por medio de este *proxy*, el ASD podrá realizar invocaciones en sentido ASD - módulo.

4.6.3 Módulo implementado: visor de alarmas.

En esta versión del sistema hay implementado un módulo que sigue esta filosofía de integración. En concreto, la misión de este módulo es la de permitir que el administrador, o un usuario normal, pueda monitorizar y controlar el sistema de gestión en su conjunto, así como poder ver, según se producen, todas las alarmas que se generan en el sistema, todo ello con una interfaz gráfica sencilla y amigable. A continuación se particularizan las explicaciones anteriores a la interfaz concreta de este módulo con el ASD con el que se relaciona, que es el gestor de configuración.

4.6.3.1 Tipo de módulo.

Concretamente, el visor de alarmas (MVA) pertenecería a la categoría de módulos con invocación de órdenes en ambos sentidos. Por lo tanto, el visor de alarmas constará de los siguientes submódulos destacables a este nivel:

- Interfaz ASD - MVA.
- Gestor de Conexión ASD - MVA.

Las otras dos partes que se mencionaban -interfaz visor de alarmas/ASD y gestor conexión visor de alarmas/ASD- estarán incluidas en el ASD, en nuestro caso, en el gestor de configuración.

En concreto, estas dos partes son las siguientes:

- Interfaz MVA - ASD.
- Gestor de Conexión MVA - ASD.

Como nota aclarativa sobre lo expuesto, decir que, por supuesto, ambas interfaces deben ser conocidas a ambos lados (visor de alarmas y gestor de configuración), pues su cometido es, precisamente, actuar como interfaces bien conocidas a ambos lados. Se mencionan en este orden para especificar dónde se implementan los métodos que soportan.

4.6.3.2 Parte del gestor de configuración.

Orden MVA -> MCF	Descripción
PDId **SDR_Instanciar_PD(** *ASDidDest, CPDId*)	Solicita la instanciación de un PDI, del tipo especificado y el ASD requerido. Se traducirá a una instrucción *Instanciar_PD*() de la API de alto nivel.
SDR_Suspender_PD(*PDId*)	Solicita la suspensión de un PDI determinado, siendo traducida a una instrucción *Suspender_PD*() de la API.
SDR_Reanudar_PD(*PDId*)	Solicita la reanudación de un PDI previamente suspendido. Es traducida a Reanudar_PD() de la API.
SDR_Terminar_PD(*PDId*)	Ordena la terminación de un PDI. Se traduce a *Terminar_PD*() en la API.
SDR_PonerCanalSalida_PD(*PDIOrig, PDIDest*)	Cambia el canal de salida de un PDI para que se enlace con el de entrada de otro. Se traduce a *PonerCanalSalida_PD*().
TablaPDI **SDR_ObtenTablaPDI**()	Obtiene la tabla de PDI durante la inicialización de la misma, o ante un fallo.
SDR_IniciarMVA(*proxyModulo, PuertoEsc*)	Inicializa el visor de alarmas en el gestor de configuración, lo da de alta en la tabla de PDI, proporciona al gestor de configuración su *proxy* y le notifica su disposición a recibir entradas.
SDR_FinalizarMVA()	Da de baja en la tabla global de PDI al visor de alarmas.

La lógica incluida en el gestor de configuración se refiere, básicamente, a la escucha del *look up* por parte del visor de alarmas y al soporte de la lógica correspondiente a las invocaciones desde el visor de alarmas. Aquí se dará soporte, pues, a la interfaz que especifica los métodos que se pueden invocar (MVA -> MCF) y la conexión del mismo.

En cuanto a las funcionalidades que proporciona el gestor de configuración al visor de alarmas, decir que, básicamente, estas tratan con el control de instancias de PDI. Desde el visor de alarmas se pueden instanciar PDI en cualquier parte del sistema, así como controlar su ejecución (suspenderlos, reanudarlos, cambiarles el canal o terminarlos). Estas órdenes tienen una correspondencia bastante cercana a la API de alto nivel para control de instancias en los ASD, lo que simplifica la tarea en cuanto a la programación de estas funcionalidades en el visor de alarmas.

Sin embargo, una característica soportada en esta interfaz de especial importancia es la de notificación de la inicialización de un visor de alarmas al gestor de configuración. Con esta orden, se le avisa de su presencia y se la pasa el *proxy* correctamente incializado para que pueda enviar órdenes en sentido inverso (gestor de configuración -> visor de alarmas). De este modo, no resulta necesario tener un registro de RMI en la máquina en la que se ejecuta el cliente ni realizar un *look up* desde el servidor.

4.6.3.3 Parte del visor de alarmas.

Podemos encontrar aquí tanto la interfaz para soportar las invocaciones desde el gestor de configuración como el gestor correspondiente, encargado de implementar la lógica necesaria para atenderlas. Las llamadas que soporta están relacionadas, básicamente, con la lógica necesaria para mantener al visor de alarmas al tanto de cualquier cambio que se produzca en los PDI que se están ejecutando en el sistema. De esta forma, el visor de alarmas puede ser notificado al momento de que se produzca un evento que le resulte relevante, sin tener que realizar interrogaciones periódicas.

Orden MCF -> MVA	Descripción
SDR_InfoPDI(*PDId, datosPDI*)	Se notifica al visor de alarmas que se ha creado un PDI en el sistema, o que han cambiado sus datos.
SDR_RemoveInfoPDI(*PDId*)	Se notifica al visor de alarmas que ha desaparecido un PDI del sistema, y que su entrada debe ser eliminada.
SDR_TablaGlobalPDI(*tablaPDI*)	Especifica una nueva tabla global de PDI obtenida a través del gestor de configuración.

El gestor de conexión entre el gestor de configuración y el visor de alarmas contendrá, pues, la implementación de esta lógica especificada en la interfaz anterior. Es más sencilla, como se puede ver, que la correspondiente para el gestor de conexión entre el visor de alarmas y el gestor de configuración.

4.6.3.4 Coherencia de las tablas de PDI en el sistema.

El visor de alarmas va a poder tener una visión global de los PDI que hay en el sistema en cada momento. Para ello, hace uso de la información contenida en la tabla global de PDI que se almacena y mantiene en el gestor de configuración.

Las primitivas del visor de alarmas para notificación de cambios de la tabla global de PDI son invocadas automáticamente por la lógica interna de gestión de las tablas. Hay que tener en mente el hecho de que, por un lado, tenemos órdenes de control sobre los PDI y, por otro lado, tenemos órdenes de mantenimiento de la coherencia de la tabla global. Cuando invocamos una orden de control, como resultado de la misma se actualiza la tabla local de PDI de uno o varios ASD, pero ahí acaba el cometido de esa orden de control. Por lo tanto, es misión de la lógica de las tablas de PDI el notificar al gestor de configuración cualquier cambio que haya en las mismas (encapsulación en la clase *HashtablePDI*), que se encargará de invocar las primitivas de bajo nivel de actualización de tabla. En el supuesto de que exista un visor de alarmas en ese momento, la actualización le será comunicada por el mismo procedimiento.

Reiterando lo que se dice en la sección pertinente del subsistema de comunicaciones, esta lógica es vital para el mantenimiento de la coherencia de las tablas -y como consecuencia directa, de los ASD-, siendo imprescindible que esta lógica, en futuras versiones, se almacene indisolublemente

con las tablas. Así, se evitarán problemas de muy difícil solución. Actualmente, la política seguida es la de encapsularla en la misma clase.

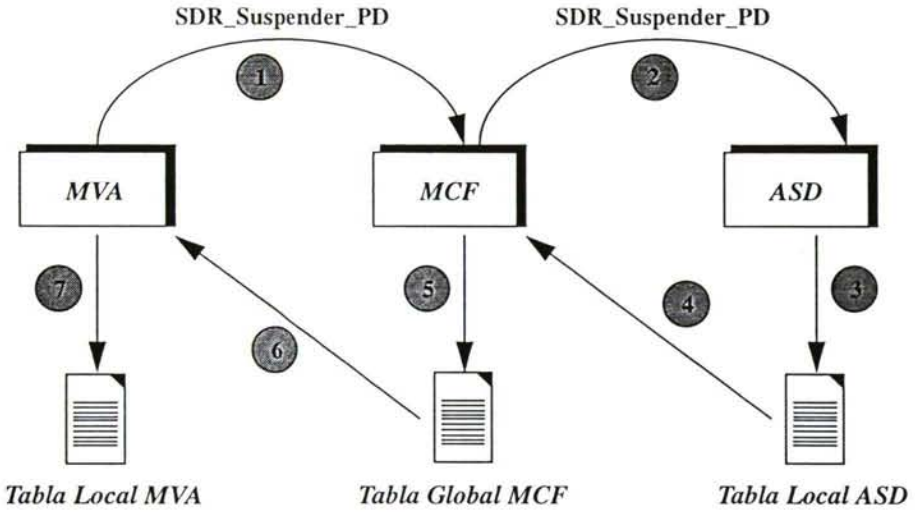

Figura 4.5. *Ejemplo de actualización de tablas.*

A modo de ilustración, en la figura podemos observar cómo se mantiene la coherencia de las tablas de PDI al producirse un cambio, fruto de una orden de control de PDI (aunque podría producirse por una terminación anormal del mismo, si un ASD así lo considera oportuno). Los pasos de la Reiterando lo que se dice en la sección pertinente del subsistema de comunicaciones, esta lógica es vital para el mantenimiento de la coherencia de las tablas -y como consecuencia directa, de los ASD-, siendo imprescindible que esta lógica, en futuras versiones, se almacene indisolublemente con las tablas. Así, se evitarán problemas de muy difícil solución. Actualmente, la política seguida es la de encapsularla en la misma clase. se explican a continuación, suponiendo que la orden proviene del visor de alarmas y va a un ASD que no sea el gestor de configuración:

- *Orden del visor de alarmas.*

  El usuario invoca una orden concreta desde el visor de alarmas para un PDI que se está ejecutando en el sistema. Esta es transmitida al gestor de configuración, para que éste se encargue de su despacho.

- *Traducción de la orden.*

  Dado que la orden es para otro ASD, la API de alto nivel la traduce a la secuencia de órdenes remotas de más bajo nivel que sean pertinentes.

- *Actualización de la tabla local.*

  El ASD destinatario, una vez ejecutada, actualiza su tabla de PDI.

- *Notificación al gestor de configuración.*

  Una vez actualizada la local, la lógica de la clase de las tablas notifica al gestor de configuración el cambio, para mantener la tabla global íntegra.

- *Actualización de la tabla global de PDI.*

  Como resultado de la notificación, se actualiza efectivamente la tabla global.

- *Notificación al visor de alarmas.*

  Si hay un visor de alarmas ejecutándose (en este caso sí), se le notifica el cambio para que lo muestre al usuario. El visor de alarmas mantiene una copia de la tabla global de PDI.

- *Actualización de la información.*

  Una vez notificado, actualiza su copia de la tabla global y refresca la información de la pantalla.

4.7 Conclusiones

La elección de Java como lenguaje de desarrollo de la lógica del sistema se revela como un acierto, puesto que su sencillez y sus características peculiares le hacen idóneo para la implementación de los componentes fundamentales del sistema.

El grado de flexibilidad y escalabilidad obtenido resulta muy elevado. La implementación de los módulos del ASD ha resultado sencilla. Además, está muy estructurada, por lo que resultará muy sencillo añadir funcionalidades nuevas a los mismos. Mecanismos como la delegación, eje central del comportamiento del sistema, o los mecanismos de planificación de procesos y gestión de órdenes entre los ASD han quedado implementados de un modo sencillo, fácilmente modificable, extensible y eficiente. En parte esto ha sido posible gracias a Java.

Otro punto interesante en lo referente a la plataforma de desarrollo es el hecho de que Java es un lenguaje en ebullición, al que se le están añadiendo funcionalidades en su API estándar constantemente, con especial mención a las comunicaciones (vitales en este desarrollo), seguridad y otros aspectos de programación general (GUI, librerías de clases especializadas, etc). Baste citar como ejemplo el hecho de que en la API estándar se proporcionan clases para la transmisión encriptada por la red, sin programación por parte del desarrollador.

El tráfico de gestión en la red, demasiado importante con arquitecturas centralizadas, se ve drásticamente reducido, puesto que la información puede ser tratada en el mismo nodo donde es producida, siendo innecesario, en muchos casos, que esta sea transmitida a quien está monitorizando -la mayoría la filtraría, pero cargaría mucho la red-. Todo lo que se ha experimentado hasta ahora demuestra que el tráfico se reduce considerablemente, puesto que el tráfico de gestión derivado del tráfico de órdenes y del mecanismo de delegación (carga dinámica de clases) es pequeño en relación al de alarmas en el sistema (es netamente superior).

Así mismo, los mecanismos para la coordinación de los diferentes módulos ASD se han revelado como eficaces y eficientes. Como mejora a la arquitectura conceptual, se ha hecho que el sistema maneje las órdenes de coordinación (primitivas) por medio de métodos remotos (RMI). De este modo, además de simplificarse su lógica, se ha podido estructurar en varios niveles de abstracción, gracias a lo cual las tareas de programación de funcionalidades adicionales se simplifican, haciendo hincapié en la flexibilidad del sistema.

5 .- *Conclusiones*

En este capítulo se realiza un resumen de las conclusiones y aportaciones más relevantes derivadas de este trabajo. En primer lugar, el estudio de diversas arquitecturas y modelos de gestión de red ha sentado las bases para determinar las características que un eficiente sistema de gestión de alarmas debe tener.

Un sistema de gestión de alarmas debe proporcionar al operador facilidades de manejo y configuración del visualizador de eventos que esté utilizando. Además, esta aplicación gráfica no debe depender de la localización de un único gestor (al menos en un entorno de fallos, en la que la ausencia de conectividad entre nodos suele ser habitual). Otra característica importante de este tipo de sistemas debe ser su capacidad multiusuario. De este modo, puede realizarse una distribución del conjunto de alarmas a gestionar en base a motivos de organización, políticos, de jerarquía, etc. Sin embargo, esta característica añade problemas de seguridad que deben ser solventados mediante mecanismos de control de acceso, encriptación y autenticación de mensajes.

Un sistema de gestión de alarmas ha de ser fácilmente ampliable mediante la incorporación de nuevos servicios y funcionalidades de gestión. Para esto, la mejor opción es el desarrollo de APIs de alto nivel que encapsulen toda la complejidad inherente a los protocolos de gestión subyacentes. Además, la incorporación de esta funcionalidad debe llevarse a cabo de un modo descentralizado de forma que no comprometa el rendimiento global del sistema, por lo que las técnicas de distribución y código móvil deben sustentar el diseño del sistema. La filosofía a seguir es la construcción de agentes de gestión genéricos, con bajo peso computacional, que puedan incorporar funcionalidad de forma dinámica. El uso de un modelo de delegación de código permite balancear la carga computacional entre los diversos nodos involucrados en la gestión de la red.

En una red de telecomunicaciones moderna coexisten gran cantidad de protocolos, cada uno de los cuales emite sus notificaciones de una forma distinta y con una semántica totalmente heterogénea. Esta cualidad complica enormemente la labor del operador de red, pues debe interpretar esta información siguiendo varios modelos de análisis. Los algoritmos que implican la correlación o análisis conjunto de alarmas resultan complejos de desarrollar. La utilización de un formato único

y estandarizado para el manejo y almacenamiento de todas las notificaciones existentes en una red es, por tanto, fundamental.

Por otro lado, resulta imprescindible la posibilidad de interacción entre el sistema de gestión desarrollado y otras aplicaciones existentes creadas con fines administrativos (pe. una aplicación desarrollada para llevar un control de las alarmas que afectan semanalmente a un determinado conjunto de dispositivos). Así, el uso de metodologías basadas en orientación a objetos posibilitan la utilización de técnicas de distribución e integración de aplicaciones como CORBA o DME.

Un sistema de gestión de fallos ha de ser, ante todo, fiable, en el sentido de continuar con la ejecución de la mayoría de sus tareas en condiciones de fallo. De nada sirve un sistema de gestión de fallos que no pueda operar durante periodos de no conectividad. La mayoría de los modelos centralizados presentan este inconveniente, pues, o bien, no se recibe información de los agentes por parte del gestor que está realizando las operaciones de gestión, o bien, no es posible invocar acciones asociadas a esa alarma para conseguir una apropiada restauración del servicio.

El uso de gestores de bases de datos comerciales para el registro de alarmas para tratamiento de históricos está muy extendido. Estos gestores posibilitan un acceso eficiente y seguro a la información de gestión y simplifican enormemente el desarrollo de aplicaciones que manejen históricos. Algoritmos que involucren el manejo de alarmas de diversos orígenes, como pe. la correlación de alarmas, encuentran en estos gestores el soporte necesario para el procesamiento eficiente de las mismas. Otra capacidad que incorporan estas aplicaciones es la posibilidad de incorporación de información estática a las alarmas. La gravedad o causa probable de un alarma puede ser determinada a partir de datos basados en la experiencia del operador, almacenados en una base de datos.

La arquitectura presentada en el capítulo 3 de esta memoria sigue todas estas recomendaciones. En cada nodo de gestión existen un proceso de baja carga computacional que puede incorporar funcionalidad dinámicamente y actuar de manera autónoma en casos de fallo o no conectividad con otros nodos del sistema. Se reduce enormemente el tráfico de gestión debido a que la información es tratada donde se genera. En un sistema de gestión de alarmas el peso de la información que generan las alarmas siempre resulta mucho mayor que el introducido por el intercambio de código entre nodos (mucho más estable temporalmente). El sistema puede ser balanceado de manera que se distribuya dinámicamente la carga global de procesamiento en base a las capacidades de los distintos nodos del sistema.

La arquitectura utiliza un formato de registro de alarmas estandarizado (ITU-T X.733) y uniforme para todo tipo de dispositivos y agentes a gestionar. También aporta un conjunto de primitivas de comunicación en forma de APIs de alto nivel, que posibilitan el control e intercambio de información transparente entre los diversos nodos de gestión. Estas dos cualidades posibilitan el desarrollo de nuevos servicios de gestión de manera rápida y sencilla (lo que mejora con la utilización de técnicas de orientación a objetos para estos desarrollos).

Las modificaciones introducidas en el modelo de delegación tradicional, posibilitan el control multiusuario del código delegado y la incorporación de mecanismos de seguridad y control de acceso a los recursos del nodo en el que se ejecuta el agente. El uso de nodos especializados, bien conocidos por todos en el sistema, posibilita el desarrollo de herramientas de construcción de servicios y en la creación de procesos cooperativos que integran información de diversos orígenes.

El acceso a gestores de bases de datos facilita el manejo y almacenamiento de alarmas.

Además el modelo soluciona el problema del control del código delegado mediante el intercambio de órdenes de control entre los diversos nodos de la red. Si un código delegado no responde o ha quedado bloqueado, el proceso delegador puede terminar su ejecución (esto no es posible en los modelos basados en *scripts*), si es el delegador el que no responde, el proceso en el que está siendo ejecutada la instancia de código puede determinar su finalización pues es el que tiene el control de ejecución sobre la misma.

La elección de Java para la implementación de esta arquitectura ha permitido alcanzar un gran grado de flexibilidad y escalabilidad. Los mecanismos de delegación, planificación de procesos y gestión de órdenes han sido implementados de manera rápida y sencilla. Además, Java incorpora nuevas funcionalidades muy útiles en este tipo de sistemas: independencia a de la plataforma, movilidad de código, librerías de métodos de encriptación, de desarrollo de GUIs, etc.

Como trabajo futuro sobre este desarrollo está la profundización en el uso de agentes inteligentes que utilicen bases de reglas para establecer dinámicamente las políticas de gestión. Actualmente estas políticas van embebidas en el código de los procesos delegados. También resultaría útil la integración del sistema en un entorno CORBA que facilite el acceso a otras aplicaciones. Existen actualmente pasarelas Java-IDL que permiten integrar el sistema desarrollado de manera automática. Sin embargo, las consideraciones de seguridad y rendimiento que introduce CORBA son motivo de interés para analizar la adecuación de este tipo de integración.

Otro aspecto interesante, en el que se está trabajando actualmente, es la introducción de algoritmos de correlación de alarmas, que permitan localizar e identificar la causa principal de un conjunto de alarmas generado por uno o más dispositivos.

Referencias

[Aidarous, 94] Salah Aidarous and Thomas Plevyak, "*Telecommunications Network Management into the 21st Century*", IEEE Press, 1994.

[ANSA, 93] "*The ANSAware 4.1 manual set*", Architecture Projects Management, Poseidon House, Castle Park, Cambridge, 1993.

[ASN, 90] ISO / IEC 8824, "*Specification of Abstract Syntax Notation One (ASN.1)*", April 1990.

[Ban, 96] Bela Ban, "*Extending CORBA for Multi-Domain Management*", IBM Zurich Research Laboratory, August 1996.

[Bharat, 96] Bharat K. and Cardelli L., "*Migratory applications*", SRC Research Report 138, Digital Equipment Corporation, 1996.

[Black, 94] Uyless Black, "*Network Management Standards: SNMP, CMIP, TMN, MIBs and Objet Libraries*", McGraw-Hill Series on Computer Communications, 1994.

[Borenstein, 94] Nathiel S. Borenstein, "*Email With a Mind Of Its Own: The Safe-Tcl Language for Enabled Mail*", ftp://ftp.fv.com/pub/code/other/safe-tcl.tar, 1994.

[Bouloutas, 92] Bouloutas A., Calo S. and Finkel A. "*Alarm Correlation and Fault Identification in Communications Networks*", IBM Technical Report, 1992.

[Brock, 94] Brockschmidt, "*Inside OLE2*", Microsoft Press, Redmond, 1994.

[Carneiro, 97a] Carneiro V.M, Muñoz, F. et al. "*Integrating SNMP and CMIP Alarm Processing in a TMN FrameWork*", V Integrated Network Management Simposium, San Diego, California, USA, 1997.

[Carneiro, 97b] Carneiro V.M et al. "*Gestión Avanzada de Redes Corporativas de Telecomunicación*", I Jornadas de Ingeniería Telemática JITEL97, Bilbao, SPAIN, Septiembre, 1997.

[Carneiro, 98a] Carneiro, V.M et al., "*A Distributed by Delegation and Multiprotocol Alarm Management System*", 7th IFIP/ICCC International Conference on Information Networks and Data Communications, Aveiro (Portugal), 15-17 June, 1998.

[Carneiro, 98b] Carneiro, V.M et al., "*Using Management by Delegation for Distributed Alarm Management*", IEEE SICON 98, Singapore, 30 June - 3 July, 1998.

[Carneiro, 98c] Carneiro, V.M et al., "*Distributed Alarm Management by Delegation for Heterogeneous and Multiprotocol Network Management*", Workshop "Distributed Computing on the Web", Rostock (Germany), 22-23 June, 1998.

[Carneiro, 98d] Carneiro V.M et al., "Alarm Management System applied to SCADA environments in Power Utilities", SBRC 98, Rio de Janeiro (Brasil), 25-29 May, 1998.

[Case, 90] Case, J.; Fedor, M.; Schoffstall, M.; Davin, J., "*A Simple Network Management Protocol (SNMP)*", RFC 1157, Network Working Group. 1990.

[Case, 93] J. Case, K. McCloghrie, M. Rose and S.Waldbusser,"*Manager-to-Manager Management Information*", RFC1451, April 1993.

[Case, 93b] Jeff D. Case and David B. Levi, "*SNMP mid-level-manager MIB*", Internet Draft, 1993.

[Case, 93c] Jeff D. Case and David B. Levi, "*SNMP script language*", Internet Draft, 1993.

[Embry, 90] Embry J., Manson P., Milham D., "*An Open Network Management Architecture: OSI/NM Forum Architecture and Concepts*", IEEE Network Magazine, pp.14-22, July 1990.

[George, 93] Jude A. George, Leslie E. Schlecht, "*The NAS Hierarchical Network Management System*", Computes Sciences Corporation, NASA Ames Research Center, 1993.

[Gold, 93] Germán Goldszmidt, "*Distributed System Management via Elastic Servers*", IEEE First International Workshop on Systems Management, Los Angeles, California, April 1993.

[Gold, 94] Germán Goldszmidt, "*On Distributed System Management*", Distributed Computing and Communication Lab, Computer Science Department, Columbia University, 1994

[Gold, 95] Germán Goldszmidt and Yechiam Yemini, "*Decentralizing Control and Intelligence in Network Management*", Proceedings of the 4th International Symposium on Integrated Network Management, Computer Science Building, Columbia University, May 1995.

[Gold, 95b] Germán Goldszmidt and Yechiam Yemini, "*Distributed Management by Delegation*", Distributed Computing and Communication Lab, Computer Science Department, Columbia University, June 1995.

[Gray, 95] Robert S. Gray, "*Agent TCL: A transportable agent system*", Department of Computer Science, Dartmouth College, Hanover, 1995

[Guerrero, 97a] Guerrero C., Carneiro, V.M. et al. "*Integrating Propietary Managed PDH Networks Using TMN-based Platforms*", First IEEE Enterprise Networking Mini-Conference (ENM97), Montreal, Canada, 1997.

[Guerrero, 97b] Guerrero C., Carneiro, V.M. et al. "*Introducing TMN in legacy networks: A step towards integrated standard management systems*", Eight IFIP/IEEE Interna-

tional Workshop Distributed Systems Operations & Management (DSOM'97), Sidney, Australia, 1997.

[Haritsa, 93] Jayant R. Haritsa, Michael O. Ball, Nicholas Roussopoulos, Anindya Datta and John S. Baras, "*MANDATE: MAnaging Networks Using DAtabase TEchnology*", IEEE Journal on Selected Areas in Communications, Vol. 11, N° 9, December 1993.

[Hughes, 93] David J. Hughes and Wu Zheng Da, "*Minerva: An Event Based Model For Extensible Network Management*", Proceedings of the INET, 1993.

[Hughes, 93b] David J. Hughes and Wu, Zheng Da, "*Minerva, An Integrated Network Management System*", Information Technology Services, Bond University, Australia, 1993.

[ISO 10040] ISO/IEC 10040, "*Systems Management Overview*".

[ISO 101164-4] ISO/IEC 101164-4, "*Alarm Reporting Function*".

[ISO 101164-5] ISO/IEC 101164-5, "*Event Report Management Function*".

[ISO 101164-6] ISO/IEC 101164-6, "*Log Control Function*".

[ISO 101164-7] ISO/IEC 101164-7, "*Security Alarm Reporting Function*".

[ISO 10165-2] ISO/IEC 10165-2, Information Technology, Open Systems Interconnection, "*Definition of Management Information*", 1991.

[ISO 10165-4] ISO/IEC 10165-4, Information Technology, Open Systems Interconnection, "*Guidelines for the Definition of Managed Objects*", 1991.

[ISO 7498-4] ISO/IEC 7498-4. Information processing systems- Open Systems Interconnection- Basic Reference Model, *"Part4: Management Framework"*. 1989.

[ISO 9595] ISO 9595 , Information Technology. Open Systems Interconnection, *"Common Management Information Services Definitions"*, 1991.

[ISO 9596] ISO 9596, Information Technology. Open Systems Interconnection,"*Common Management Information Protocol (CMIP)*", 1991.

[ITU-T X720] ITU-T X720, Information Processing - Open System Interconnection - "*Structure of Management Information: Management Information Model*", 1988.

[ITU-T X721] ITU-T X721, Information Technology - Open System Interconnection - "*Structure of Management Information: Definition of Management Information*", 1992.

[ITU-T X722] ITU-T X722, Information Processing - Open System Interconnection - "*The Directory: Overview of Concepts, Models and Service*", 1988.

[ITU-T X733] ITU-T X733, Tecnología de la Información - Interconexión de Sistemas Abiertos - "*Gestión de Sistemas: Función Señaladora de Alarmas*", 1992.

[Jakobson, 93] Gabriel Jakobson and Mark D. Weissman, "*Alarm Correlation*", IEEE Network, pp. 52-59, November 1993.

[Jander, 93] M. Jander, "*Midlevel Managers Ease SNMP Information Overload*", Data Communications, November 1993.

[JAVA, 95] Sun Microsystems Inc."*The Java Language Environment: A White Paper*", 1995.

[Kooijman, 94] Kooijman R,. Van Orscot, J. and Wiesse, D,"*RMON implementation issues for managers and agents*", Draft request for comments, Delft University of Technology, 1994.

[Kooijman, 95] R. Kooijman, "*Divide and conquer in network management using event-driven network area agents*", Technical University of Delft, The Netherlands, May 1995.

[Leinwand, 96] Allan Leinwand and Karen Fang Conroy, "*Network Management: A Practical Perspective*", 2nd Edition, Addison-Wesley, 1996.

[Lillian, 89] Lillian N. Cassel, Graig Patridge, and Jil Westcott, "*Network Management Architectures and Protocols: Problems and Approaches*", IEEE JSAC, Vol. 7, no. 7, Sept. 89.

[Liskov, 88] Barbara Liskov and Liuba Shrira. "*Promises: Linguistic Support for Efficient Asynchronous Procedure Calls in Distributed Systems*", ACM SIGPLAN, pages 22-24, Atlanta, GA, June 1988.

[Moffett, 91] Moffett J., Sloman M., "*The Representation of Policies as System Objects*", Proc. Conf. on Organisational Computer Systems (COCS 91), Atlanta USA, Nov 1991.

[Moffett, 93] Jonathan D. Moffett and Morris Sloman, "*User and Mechanism of Distributed Systems Management*", IEE/IOP/BCS Distributed Systems Engineering, Vol. 1, N°. 1, August 1993.

[Monsouri, 93] Monsouri-Sanani M, Sloman M., "*Monitoring Distributed Systems (A Survey)*", Imperial College Research Report, April 1993.

[Ohta, 97] Kohei Ohta, Takumi Mori, Nei Kato, Hideaki Sone, Glenn Mansfield and Yoshiaki Nemoto, "*Divide and Conquer Technique for Network Faul Management*", Graduate School of Information Science, Tohoku University, 1997

[OMG 91.12.1] OMG Document N° 91.12.1, "*The Common Object Request Broker: architecture and specification*", Dec 1991.

[OSF, 92] Open Software Foundation, "*The OSF Distributed Management Environment Architecture (DME)*", May 1992.

[Ousterhout, 94] John K. Ousterhout, "*Tcl and the Tk Toolkit*", Addison-Wessley, 1994.

[Pavlou, 93] Pavlou G., S. Bhatti and G. Knight, "*Automating the OSI to Internet Management Conversion Using an Object-Oriented Platform*", IFIP Conference on LAN/MAN Management, Paris, 04/93.

[Pavlou, 93b] Pavlou G., "*Implementing OSI Management*", 3rd IFIP/IEEE ISINM, San Francisco, 4/93, UCL Research Note 94/74.

[Pavlou, 95] George Pavlou, Kevin McCarthy, Saleem Bhatti, Graham Knight, "*The OSIMIS Platform: Making OSI Management Simple*", Department of Computer Science, University College London, 1995.

[RMI, 96] Sun Microsystems Inc."*Java Remote Method Invocation Specification*", November 1996.

[Rose, 90] M. T. Rose, "*The Open Book, A Practical Perspective on OSI*", Englewood Cliffs, NJ: Prentice Hall, 1990.

[Rose, 90b] M. Rose and K. McCloghrie, "*Structure and Identification of Management Information for TCP/IP-based Internets*", RFC 1155, Mayo 1990

[Rose, 91] Rose M.T., J.P. Onions, C.J. Robbins, "*The ISO Development Environment User's Manual version 7.0*", PSI Inc./X-Tel Services Ltd., 7/91.

[RPC, 84] A.D. Birrel and B.J. Nelson, "*Implementing Remote Procedure Calls*", ACM Transactions on Computer Systems, 2:39 - 59, February 1984.

[Schade, 96] Andreas Schade, "*An Event Framework for CORBA-Based Monitoring and Management Systems*", IBM Zurich Research Laboratory, 1996.

[Schon, 95] J. Schonwalder, H Langendorfer, "*TCL extensions for Network Management Applications*", Tcl/Tk Workshop, July 1995.

[Siegl, 94] Manfred R. Siegl and Georg Trausmuth, "*Hierarchical Network Management: A Concept and its Prototype in SNMPv2*", University of Technology, Vienna, Austria, 1994

[Sloman, 89] Sloman M.S. & Moffett J.D., "*Domain Management for Distributed Systems*", Integrated Network Mangement pp. 505-516, North Holland, 1989.

[Sloman, 93] Sloman M., Magee J, Twidle K. and Kramer J., "*An Architecture of Managing Distributed Systems*", Fourth IEEE Workshop on Future Trends of Distributed ComputingSystems, IEEE Computer Society Press, pp. 40-46, Lisbon, Portugal, Sep 22-24, 1993.

[Sloman, 94] Morris Sloman, "*Network and Distributed Systems Management*", Addison-Wesley Publishing Company. 1994.

[Stallings,93] William Stallings, "*SNMP, SNMPv2 and CMIP - The Practical Guide To Network Management Standars*", Addison Wesley, 1993.

[Stama, 95] F. Stamatelopoulos, T. Chiotis and F. Maglaris, "*A Scalable, Platform-Based Architecture for Multiple Domain Network Management*", Department of Electrical and Computer Engineering, National Technical University of Athens, 1995.

[Stama, 95b] F. Stamatelopoulos, N. Roussopoulos and B. Maglaris, "", Engineer Conference Networld´95, March 95.

[Stamos, 90] James W. Stamos and David K. Gifford, "*Remote Evaluation*", ACM Transactions on Programming Languages and Systems, 12(4):537-565, October 1990.

[Stroustrup, 86] Stroustrup B., "*The C++ Programming Language*", Addison-Wesley, Reading, MA, 1986.

[Terplan, 92] K. Terplan, "*Communication Network Management*", Englewood Cliffs, NJ: Prentice Hall, 1992.

[Thomas, 95] Thomas, R, "*Interoperable RMON? Plug and Pray*", Data Communications, May, 1995.

[TINA, 95] Telecommunications Information Networking Architecture Consortium, "*Overall Concepts and Principles of TINA*", Version 1.0, Feb 17 1995.

[Vassila, 95] Anastasia Vassila and Graham J. Knight, "*Introducing Active Managed Objects for Effective and Autonomous Distributed Management*", University College London, UK, 1995.

[Wald, 95] Waldbusser, S., "*Remote Network Monitoring Management Information Base*", Request for Comments 1757, Internet Engineering Task Force, 1995.

[Williamson, 96] Bradley Williamson and Craig Farrel, "*Distributed Management using the Remote Monitoring Management Information Base an extensible agents*", School of Computing, Curtin University of Technology, Perth, Western Australia, September 1996.

[White, 94] James E White, "*Telescript Technology: The Foundation for the Electronic Marketplace*", General Magic White Paper, 1994.

[XOpen, 92] X/Open, "*OSI-Abstract-Data Manipulation and Management Protocols Specification*", 1/92.

[Znaty, 95] Simon Znaty, Michel Lion, Jean-Pierre Hubaux, "*DEAL: A Delegated Agent Language for Developing Network Management Functions*", Swiss Federal Institute of Technology, 1995.

Apéndice A: Guía de usuario

En este manual de usuario se muestra el modo de utilizar el visor de alarmas desarrollado para integrarlo como módulo de nivel superior del gestor de configuración. Se detalla cada una de las distintas opciones que se encuentran disponibles en el visor.

A.1. Arranque de la aplicación

Es posible lanzar el programa como un *applet* o como una aplicación *standalone*.

En el primer caso necesitamos un navegador con soporte para *applets* de Java, igual o superior a la versión 1.1 del JDK. Por ejemplo, el Netscape 4.04 o incluso el propio *appeltviewer* incluido en el JDK1.1. El fichero que hay que cargar se llama SIGABAD.htm el cual se encarga de lanzar la aplicación.

En el segundo caso, es necesario utilizar el compilador de Java incluido en el JDK1.1. La línea que hay que teclear para que arranque la aplicación es: java SIGABAD

En ambos casos, hay que definir la variable de entorno CLASSPATH para que el intérprete de Java sepa dónde buscar los ficheros *.class, que están comprimidos en formato ZIP y no obtener así un posible NotDefFoundClass. Hay que tener en cuenta que para que la aplicación funcione, es necesario que el Gestor de Configuración esté ya lanzado (ver capítulo 4, para más detalles).

A.2. Ventana principal del Módulo de Visualización de Alarmas

La ventana principal del Módulo de Visualización de Alarmas (MVA) se muestra en pantalla una vez que se inicializa la aplicación (ver figura A.1).

Figura A.1. *Ventana principal del MVA*

Esta ventana consta de cuatro partes bien diferenciadas:

- *Menús desplegables.*
- *Panel de botones de acceso directo.*
- *Panel de visualización de alarmas.*
- *Ventana inferior de información.*

A.2.1. Menús desplegables

Los menús desplegables se encuentran en la barra de menús, en la parte superior de la ventana principal. Al igual que en los GUI's estándar, en la parte derecha de esta barra se encuentra el menú de "Ayuda", y en la izquierda el menú "Archivo" que se detalla a continuación.

A.2.1.1. Archivo

En el menú archivo nos encontramos con la opción "**Salir**", que además es la única de este menú. Como su propio nombre indica, nos permite abandonar el programa. En futuras versiones se ampliará este menú con opciones de guardado y recuperación de datos, configuración de la interfaz, etc.

A.2.1.2. Ver

Desde aquí podemos configurar el panel de alarmas. Se ofrecen las opciones de limpiar el panel y de especificar qué atributos se desean ver para las alarmas mostradas.

La opción "**Limpiar panel**" borra las alarmas del panel de visualización. No las borra de la base de datos de alarmas, solamente de la pantalla.

La opción "**Atributos...**" abre un panel de botones tipo conmutador (*toggle*) en el que se puede activar o desactivar cada uno de ellos (figura A.2). Estos atributos son un subconjunto de los definidos en la Recomendación X.721 ISO/IEC 10165-2 para el registro de notificaciones y se muestran a continuación:

- Ejemplar de Objeto Gestionado (managedObjectInstance)
- Causa Probable (probableCause)
- Acciones de Reparación Propuestas (proposedRepairActions)
- Atributos Monitorizados (monitoredAttributes)
- Problemas Específicos (specificProblems)
- Gravedad Percibida(perceivedSeverity)
- Texto Adicional (additionalText)
- Tipo de Evento(eventType)
- Información de Umbral (thresholdInfo)

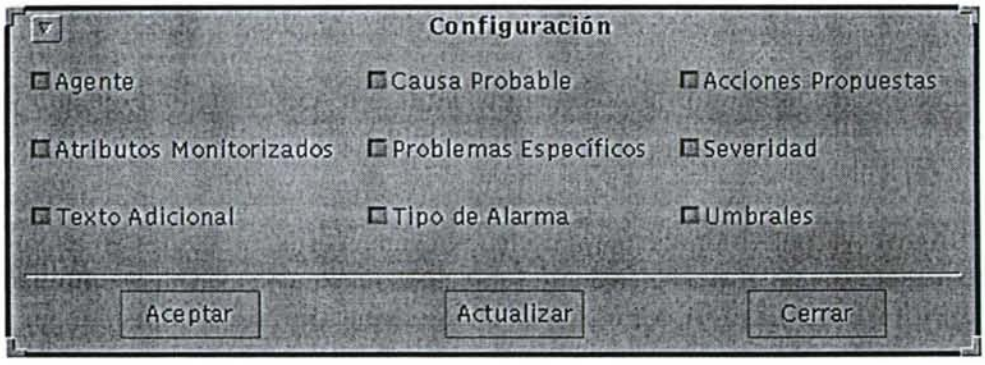

Figura A.2. *Panel de Configuración de Atributos Visibles*

No es necesario establecer los atributos a monitorizar antes de empezar la recepción de alarmas. Esta opción puede ser utilizada en cualquier momento, por lo que podemos estar recibiendo alarmas y al mismo tiempo cambiar el número de atributos que queremos ver. Independientemente de si se muestra o no un determinado atributo, la posición que ocupa respecto a los demás es siempre la misma. Inicialmente se muestran por defecto todos los atributos.

En el panel principal no se muestra el contenido completo de cada atributo, sólo un detalle de su valor. Para ver la información completa de cada atributo, se deberá usar la opción "Detalles" del menú "Alarmas", descrita a continuación.

A.2.1.3. Alarmas

Desde este menú podemos visualizar los atributos de una alarma en detalle, borrar una alarma determinada o borrar todas las alarmas que se hayan almacenado en el sistema.

La opción "**Detalles...**" abre una ventana (figura A.3) en la que aparecen todos los atributos de la alarma que se ha seleccionado en el panel principal. Cada uno de estos atributos muestra su contenido completo.

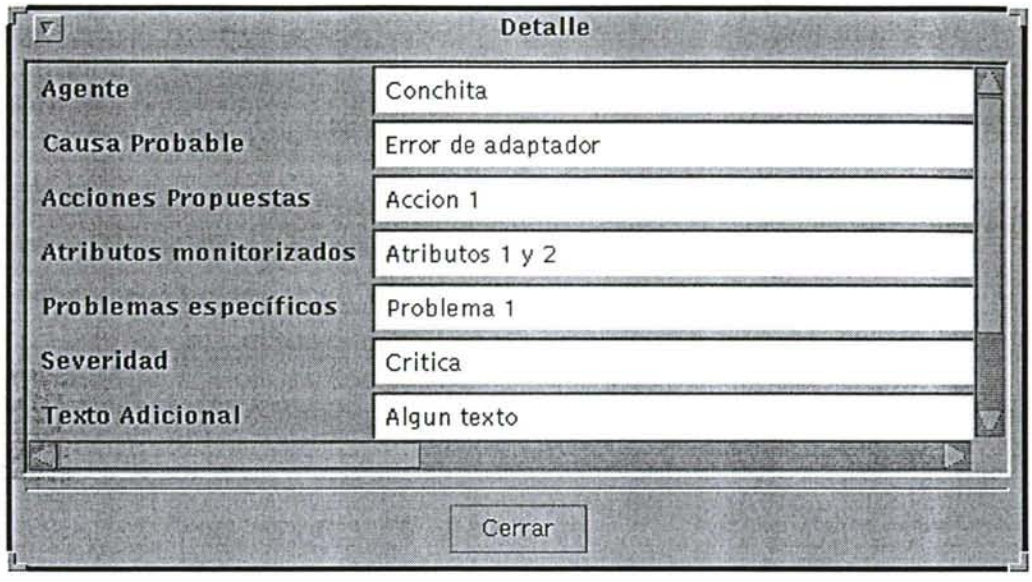

Figura A.3. *Ventana de visualización de detalles de una alarma*

Si mientras se está viendo la información de una alarma, se selecciona otra distinta y se vuelve a usar la opción "Detalles...", los datos que contiene la ventana detalles se actualizarán a los de la nueva alarma seleccionada.

La opción "**Borrar Actual**" borra la alarma seleccionada en esos momentos en el panel principal. Este borrado no solo se realiza a efectos de presentación si no que la alarma es eliminada realmente del árbol de gestión de la plataforma, no siendo accesible a partir de ese momento por ninguna otra aplicación que esté conectada a la misma.

La opción "**Borrar Todas**" se comporta de forma similar a la anterior pero en este caso se borran todas las alarmas que se estén viendo en el panel de visualización de alarmas. Previo al borrado, se muestra una ventana de confirmación (figura A.4) para evitar el borrado accidental de información.

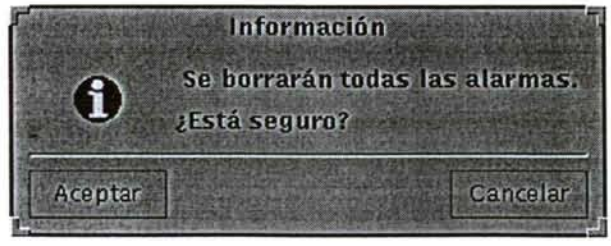

Figura A.4. *Ventana de confirmación para el borrado de alarmas*

A.2.1.4. Histórico

Desde este menú podemos ver todas las alarmas que se han producido en el sistema y que se han ido almacenando en un fichero de registro de alarmas. Este fichero es un histórico de las alarmas

que se han producido. Cuando se hace uso de las opciones de borrado, detalladas anteriormente, los cambios aparecen reflejados en este fichero.

La opción "**Ver alarmas...**" es la que nos permite visualizar estas alarmas en una ventana similar a la que se emplea en la ventana principal de SIGABAD (figura A.5). En este caso, se muestran todos los atributos de cada una de las alarmas. Desde esta ventana podemos pulsar el botón "Detalles" que nos permite ver de un modo pormenorizado la alarma que se haya seleccionado (figura A.3), de la misma manera que se hacía antes desde la ventana principal.

Figura A.5. *Ventana de análisis de alarmas de la plataforma*

A.2.1.5. Configuración

Desde este menú podemos realizar la configuración de los agentes (procesos delegados) en la red. Consta de una única entrada, "**Configurar procesos**" que abre una ventana como la que se muestra en la figura A.6. Esta ventana consta de dos botones, "Cerrar" e "Instanciar". El primero de ellos cierra la ventana de configuración y regresa a la pantalla principal. El botón de instaciar nos permite acceder a la ventana de "Detalles de Procesos", desde la cual realizaremos la configuración.

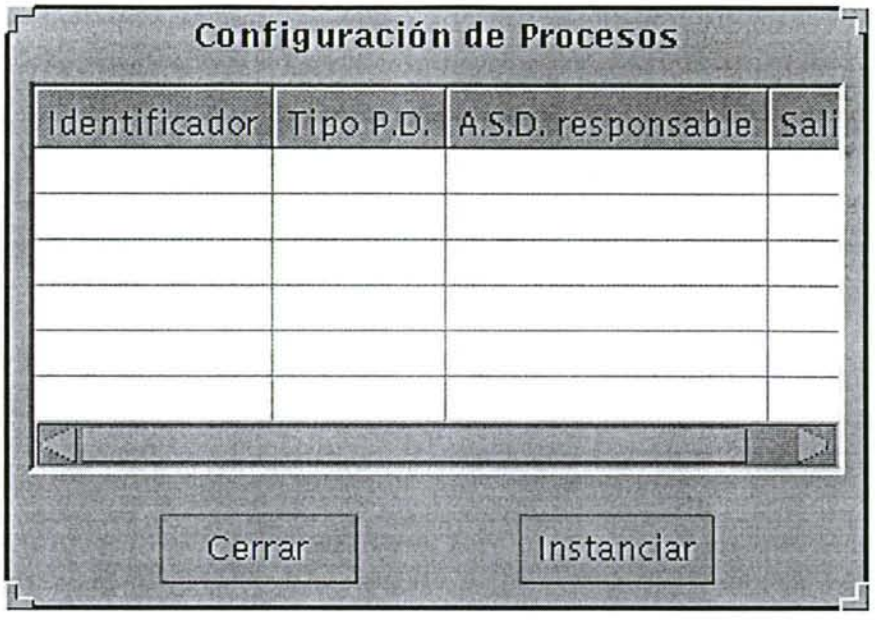

Figura A.6. *Ventana de configuración de procesos*

En la ventana de "Configuración de Procesos" existe un panel central en el que se nos muestra la siguiente información:

Campo	Significado
Identificador	Identificador de la instancia del proceso delegado. Es un número separado por guiones cuyo significado es el siguiente: 1ercampo: ASD que solicito la instanciación. 2º campo: MOC responsable del control del PD. 3ercampo: Clase a la que pertenece la instancia. 4º campo: Instancia del PD
Tipo P.D.	Clase de Proceso Delegado a la que pertenece la instancia (Generador de "traps", filtro,...)
A.S.D. Responsable	Es el A.S.D. que solicitó la delegación de la instancia
Salida	Es el A.S.D. que está "acogiendo" a la instancia.

La ventana de "Configuración de procesos" no es editable. Para poder realizar realmente la configuración accedemos a la ventana de "Detalles de procesos" (figura A.7).

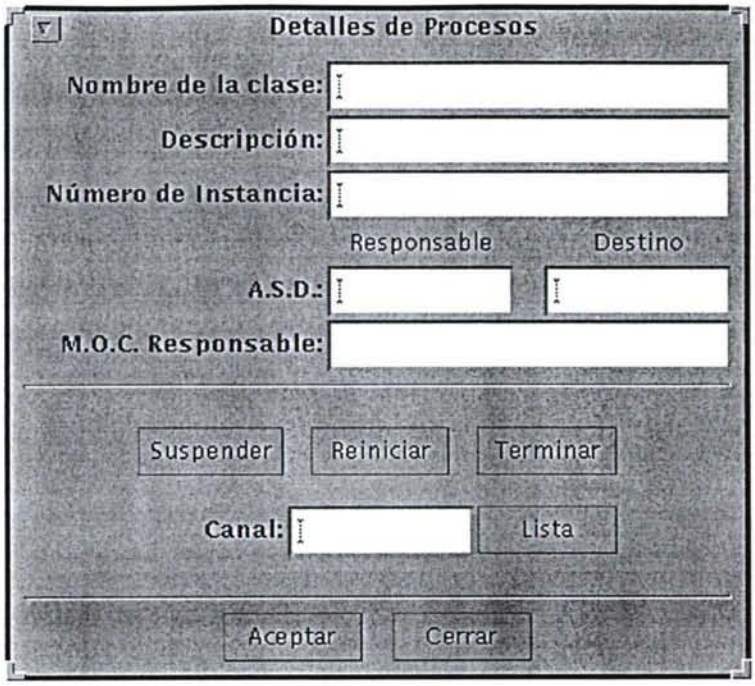

Figura A.7. *Ventana "Detalles de Procesos"*

Esta ventana tiene tres partes bien diferenciadas que detallamos a continuación:

- *Panel de identificación*
- *Panel de configuración*
- *Botones "Aceptar" y "Cerrar"*

En esta versión de SIGABAD no existen módulos de control.

En el panel de configuración de la ventana "Detalles de Procesos" nos encontramos con cuatro botones y un campo de texto, cuyo significado es el siguiente:

Botón	Acción
Suspender	Suspende la ejecución de una instancia
Reiniciar	Reanuda la ejecución de una instancia que previamente ha sido suspendida.
Terminar	Finaliza la ejecución de una instancia.
Lista	Muestra una lista desplegable con todos los canales de salida disponibles para conectar una instancia con otra. Al seleccionar un campo de la lista desplegable, aparece en la caja de texto "Canal"

Las primitivas, a nivel interno, que se generan al pulsar cada uno de estos botones, están especificadas en la estructura interna del MVA.

Finalmente, el botón de "Aceptar" realiza una instanciación del proceso delegado en cuestión, con los datos que se hayan especificado, y el botón cerrar nos devuelve a la ventana de "Configuración de Procesos".

A.2.1.6. Ayuda

En esta versión de la aplicación no se encuentra integrada la ayuda en línea. Este menú dispone sólo de una entrada, "**Sobre SIGABAD...**" que hace aparecer un cuadro de diálogo (figura A.8) en el que se muestra información general sobre la aplicación, versión y fecha de realización del mismo.

En futuras versiones, se incluirá una ayuda en línea, con opciones de búsqueda de palabras clave, etc.

Figura A.8. *Ventana de información sobre la aplicación*

A.2.2. Panel de botones de acceso directo

Debajo de la barra de menús se encuentran una serie de botones de acceso rápido (figura A.9). Cada uno de estos botones representa alguna opción de las comentadas anteriormente en los menús. La utilidad de estos botones es la de permitir acceder de un modo más rápido a opciones que se realizan con cierta frecuencia, sin tener que abrir el menú desplegable y escoger la opción correspondiente.

Figura A.9. *Panel de botones de acceso rápido*

Dependiendo del estado de la aplicación algunos de estos botones pueden estar desactivados.

La asignación entre cada uno de los botones y la opción del menú correspondiente se muestra en la siguiente tabla:

Opción del Menu	Botón
Histórico/Ver Histórico	Histórico
Ver/Atributos...	Config.
Alarmas/Detalles...	Detalle
Alarmas/Borrar Actual	Borrar
Archivo/Salir	Salir

Tabla 7. *Significado de Botones de acceso rápido*

Existe un botón adicional que no se corresponde con ninguna opción de los menús desplegables. Este botón es el de "**Pausa/Reiniciar**". Cuando arrancamos la aplicación, el programa está listo para recibir alarmas. Si por cualquier motivo queremos ver las alarmas que han llegado, sin que nuevas alarmas desplacen a las que ya están representadas en el panel, podemos pulsar el botón de "Pausa". Las alarmas que lleguen después de haber pulsado este botón no se visualizarán, pero tampoco se perderán, ya que cuando volvamos a pulsar el botón (que en este momento tendrá la etiqueta "Reiniciar") se mostrarán en pantalla.

A.2.3. Panel de alarmas

Se encuentra en el centro de la interfaz principal y ocupa la mayor parte de la misma. En ella se muestra una información abreviada sobre las nuevas alarmas que sean recibidas por la aplicación, a medida que éstas vayan llegando. Las alarmas se ordenan cronológicamente de arriba a abajo, colocando las más recientes en la parte inferior.

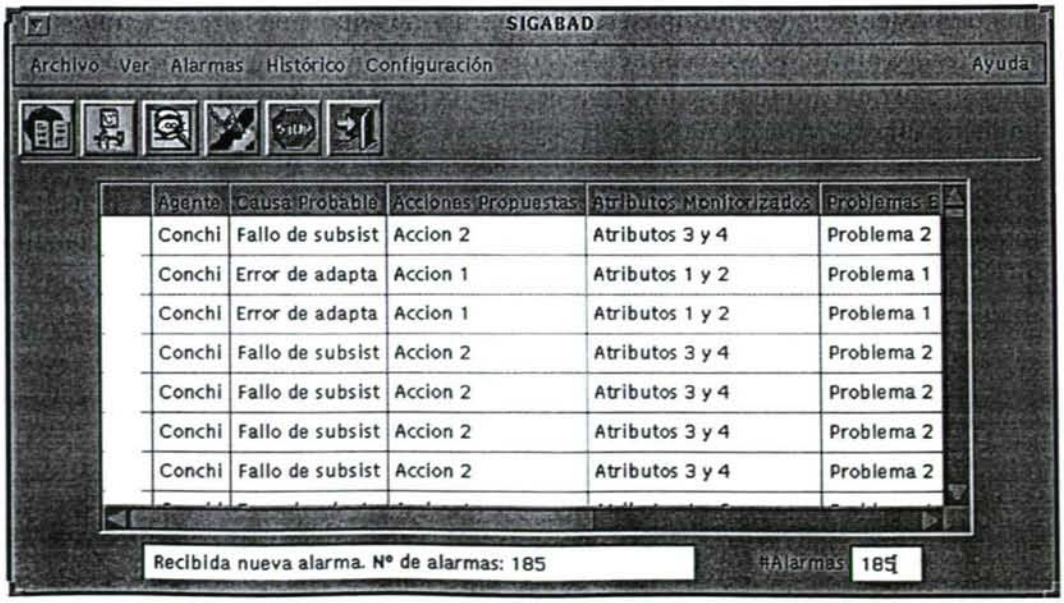

Figura A.10. *Panel de visualización de alarmas*

En su parte superior se indican los nombres de los atributos que se muestran para cada alarma del panel. Esta lista de atributos es configurable, con lo que se permite seleccionar solo los parámetros de interés de cada alarma.

Además de los atributos especificados, en la parte izquierda de cada entrada del panel, se encuentra un indicador de su gravedad percibida mediante un código de colores que se indica en la siguiente tabla:

Severidad	Color
Indeterminada	Azul Oscuro
Crítica	Roja
Mayor	Naranja
Menor	Amarilla
Aviso	Amarilla
Eliminada	Gris

Tabla 8. *Asignación de colores a severidades*

El panel dispone de barras de desplazamiento horizontal y vertical para ver todas las alarmas recibidas y todos sus atributos.

El panel permite seleccionar una única alarma, bien para borrarla o bien para examinar detalles.

A.2.4. Ventana de estado

Bajo el panel de presentación de alarmas, se encuentran la ventana en la que se muestran la mayor parte de los mensajes emitidos por la aplicación que sean de interés para el operador. Se confirman las operaciones realizadas, o bien se informa de que ha habido algún error mientras se trataba de realizar alguna de ellas.

Se compone de un panel de presentación de texto, en el que se muestran los mensajes, y de un indicador en los que se muestran el número de alarmas que hay en el panel de visualización de alarmas en esos momentos y el identificador de la que está seleccionada en esos momentos, respectivamente.

Apéndice B: Glosario

API (Aplication Programming Interface
ASD (Arquitectura para el Soporte de la Delegación)
ASDid (Identificador de proceso elástico)
ASN.1 (Abstract Syntax Notation One)
BER (Basic Encoded Rules)
CDid (Identificador de Código Delegable)
CMIP (Common Management Information Protocol)
CMISE (Common Management Information Service Element)
CPU (Central Process Unit)
DBMS (Data Base Management System)
DES (Data Encription Standard)
DN (Distinguishing Name)
EFD (Event Forward Discriminator)
EGP (External Gateway Protocol)
GUI (Graphic User Interface)
HNMS (Hierarchical Network Management System)
IP (Internet Protocol)
ISO (International Standard Organization)
LAN (Local Area Network)
MAR (Gestor de Adaptadores)
MbD (Management by Delagation)
MCF (Gestor de Configuración)
MCM (Módulo de Comunicaciones)
MD5 (Message Digest 5)
MGP (Módulo Gestor de Procesos Delegados)
MI (Management Information)

MIB (Management Information Base)
MIR (Gestor de Interrogadores)
MIT (Management Information Tree)
MO (Managed Object)
MOC (Módulo de Control)
MOM (Manager of Managers)
MPR (Módulo Principal)
MRP (Módulo Repositorio)
MVA (Visor de Alarmas)
NMP (Network Management Procedure)
OI (Object Identifier)
OID (Object Identifier)
OSI (Open System Interconection)
PDI (Instancia de Proceso Delegado)
PDId (Identificador de instancia de proceso delegado)
PDU (Protocol Data Unit)
PE (Proceso Elástico)
QoS (Quality of Service)
RDN (Relative Distinguishing Name)
RMON (Remote Monitoring)
RPC (Remote Procedure Calls)
SAP (Service Access Point)
SDR (Servicio de Delegación Remota)
SMF (Service Management Function)
SMI (Structure of Management Information)
SNMP (Simple Network Management Protocol)
SQL (Structured Query Language)
UDP (User Datagram Protocol)
WAN (Wide Area Network)

Indice